

SEPTEMBER 1987

COMPUTER



EXPLORING
COMPUTER
CONCEPTS

Hypertext
RISC • CISC
Software Productivity
Formalism in Program Design



THE COMPUTER SOCIETY



THE AMERICAN NUCLEAR SOCIETY
ELECTRICAL AND
ELECTRONIC ENGINEERS, INC.

Reader Service Number 1

SIMULATION TEACHING BREAKTHROUGH

BEFORE

TEST NETS	0	SSL(1)	-1.67	X	-1.73	SSL(2)	4.77	Y	4.82	Y1	4.00
TEST NETS	0	SSL(1)	-1.67	X	-1.70	SSL(2)	4.77	Y	4.00	Y1	4.00
TEST NETS	0	SSL(1)	-1.70	X	-1.73	SSL(2)	4.00	Y	4.82	Y1	4.00
PAGE 6											
TIME		X		Y		SPEED		DATA		RT BT	DT BT
TEST NETS	0	SSL(1)	-1.73	X	-1.83	SSL(2)	4.82	Y	4.90	Y1	4.00
TEST NETS	0	SSL(1)	-1.73	X	-1.70	SSL(2)	4.82	Y	4.06	Y1	4.00
TEST NETS	0	SSL(1)	-1.73	X	-1.75	SSL(2)	4.82	Y	4.04	Y1	4.00
TEST NETS	0	SSL(1)	-1.75	X	-1.70	SSL(2)	4.04	Y	4.06	Y1	4.00
TEST NETS	0	SSL(1)	-1.70	X	-1.85	SSL(2)	4.06	Y	4.32	Y1	4.00
TEST NETS	0	SSL(1)	-1.70	X	-1.81	SSL(2)	4.06	Y	4.69	Y1	4.00
TEST NETS	0	SSL(1)	-1.70	X	-1.79	SSL(2)	4.06	Y	4.80	Y1	4.00
TEST NETS	0	SSL(1)	-1.79	X	-1.81	SSL(2)	4.80	Y	4.89	Y1	4.00
TEST NETS	0	SSL(1)	-1.81	X	-1.87	SSL(2)	4.89	Y	4.94	Y1	4.00
TEST NETS	0	SSL(1)	-1.81	X	-1.94	SSL(2)	4.89	Y	4.92	Y1	4.00
TEST NETS	0	SSL(1)	-1.84	X	-1.87	SSL(2)	4.92	Y	4.94	Y1	4.00
TEST NETS	0	SSL(1)	-1.87	X	-1.90	SSL(2)	4.94	Y	4.96	Y1	4.00
TEST NETS	0	SSL(1)	-1.90	X	-1.93	SSL(2)	4.96	Y	4.90	Y1	4.00
.130000 -1.93849 4.98318 051.98518 .04238 -40.0560 36.8218											

Difficult to understand results

AFTER



You see an animated picture of the system under study--any application

Now for universities PC AT SIMSCRIPT II.5 with SIMANIMATION new simulation teaching package--provides animated results

With PC SIMSCRIPT II.5 and new animation, simulation results are easy-to-understand--moving pictures, histograms, pie charts and plots.

Because simulation results are understood, you can concentrate on teaching simulation concepts.

Simulation animation simplified

SIMSCRIPT II.5 gives you a compact English-like language. The simulation program reads like a description of the system under study. Animation is easy.

Your students' model development, checkout, modification and enhancement are greatly simplified.

Many successful applications

SIMSCRIPT II.5® is a well established, standardized, and widely used language with proven software support.

Typical applications include: manufacturing, communications, logistics, transportation and military planning.

Low cost for universities

CACI recognizes the benefits of having its state-of-the-art systems used by the universities.

For that reason we make the PC SIMSCRIPT II.5 teaching package available to universities for only a low distribution charge.

The package includes training, support, complete documentation, and sample models. Everything you need to conveniently teach simulation analysis.

Act now-limited offer

This offer is limited to 1,500 universities, and 1,273 have already signed up. Don't be disappointed--call today.

For immediate information

Call Rick Crawford at (619) 457-9681. In the UK, call Steve Wombell on (01) 940-3606.

With PC SIMSCRIPT II.5 your students get results sooner and the results are better understood.

Rush information on the special simulation teaching package for universities only.

Everything you need to teach PC SIMSCRIPT II.5 with animation.

- ☐ Also send information about NETWORK II.5. Network analysis with no programming.
- ☐ Also send information about SIMFACTORY. Factory planning with no programming.

Name

Organization

Address

City State Zip

Telephone

Computer Operating System

Return to: IEEE COMP

CACI
3344 North Torrey Pines Court
La Jolla, California 92037
Or, better yet, call Rick Crawford at (619) 457-9681

In the UK
CACI
26 The Quadrant
Richmond, Surrey, England TW9 1DL
Call Steve Wombell on (01) 940-3606

COMPUTER

September 1987

Published by the Computer Society of the IEEE

Vol. 20, No. 9

SURVEYS & TUTORIALS

17 Hypertext: An Introduction and Survey

Jeff Conklin

Hypertext systems feature machine-supported links—both within and between documents—that open exciting new possibilities for using the computer as a communication and thinking tool.

43 Improving Software Productivity

Barry W. Boehm

By 1995, a 20 percent improvement in software productivity will be worth \$90 billion worldwide. Clearly, the measures outlined in this article are worth the effort.

59 A Survey of RISC Processors and Computers of the Mid-1980s

Charles E. Givarc and Veljko M. Milutinović

We briefly survey modern RISC architectures, illustrating common features, range of applications, implementation technologies, and some unique characteristics of today's RISC machines.

85 An Overview of Some Formal Methods for Program Design

C.A.R. Hoare

The design of a small program, like that of a large system, requires a variety of formal methods and notations, related by mathematical reasoning.

SPECIAL FEATURE

71 And Now a Case for More Complex Instruction Sets

Michael J. Flynn, Chad L. Mitchell, and Johannes M. Mulder

Using a computer architecture simulation platform, we can perform instruction set tradeoffs with a common optimizing compiler and workload.

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the Computer Society of the IEEE. IEEE Headquarters: 345 East 47th St., New York, NY 10017; Computer Society of the IEEE, Publications Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-8380. Annual subscription included in society member dues. Nonmember prices: available upon request. Single copy prices: members \$7.50; nonmembers \$15.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, New York, and at additional mailing offices.

Copyright and reprint permissions: Copyright © 1987 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons: (1) those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress Street, Salem, MA 01970. (2) pre-1978 articles without fee. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to True Seaborn, Editor and Publisher, *Computer*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720. All rights reserved. Copyright © 1987 by The Institute of Electrical and Electronics Engineers, Inc.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space considerations. *Computer* subscribes to the Computer Press Association's code of professional ethics.

 BPA

DEPARTMENTS

- 6 President's Message**
The benefits of belonging
- 10 Editor-in-Chief's Message**
Changes—obvious and subtle
- 94 Standards**
Point/Counterpoint: IEEE Trial-Use Standards
- 98 Update**
- 99 Computer Society News**
CS calls for tax section repeal; computer bridge contest;
Microprocessor/Microcomputer TC; Security and Privacy TC
- 103 New Product Reviews**
Interactive videodisc technology
- 107 New Products**
- 114 IC Announcements**
- 115 Microsystem Announcements**
- 118 Conferences**
FJCC offers broad program; AFIPS votes to continue NCC
- 119 Calendar**
- 121 Call for Papers**
- 124 Book Reviews**
- 127 New Literature**



On the cover

A dazzling chamber of light awaits our cover diver as she enters a lava tube cave off the coast of the Hawaiian island of Lanai. Discoveries of another sort await readers of this month's surveys and tutorials, which explore some of today's more intriguing computer concepts.

Photo: Tom Stack/Tom Stack & Associates

Design: Jay Simpson

In the next issue

Spatial Light Modulator
Interconnected Computers; Optical
Techniques in Knowledge Base
Machines; Abstract Data Type
Components; Array Computer Algo-
rithm for Query Processing;
Distributed Software Engineering

Career Opportunities, 122; Advertiser/Product Index, 128;
Reader Service Card, 128A; Change-of-Address Form, 101;
Membership Application, 9; Membership Information, Cover 111

COMPUTER

10662 Los Vaqueros Circle, Los Alamitos, CA 90720

Editor-in-Chief: Bruce D. Shriver, IBM

Editorial Board

Dharma P. Agrawal, North Carolina State University
Scott Davidson, AT&T
Richard Eckhouse, Moco Inc.
Gideon Frieder, University of Michigan
Edmund Gallizzi, Eckerd College
John Gurd, University of Manchester
James H. Haynes, University of California, Santa Cruz
Franklin F. Kuo, SRI International
Wiley McKinzie, Rochester Institute of Technology
K. M. Mohiuddin, IBM Almaden Research Lab.
William Riddle, Software Productivity Consortium
Sallie Sheppard, Texas A&M University
Harry Strong, Mitre Corporation
Helen M. Wood, National Bureau of Standards

Submissions:

Send six double-spaced copies of articles and special-issue proposals to Bruce D. Shriver, IBM T.J. Watson Research Center, PO Box 704, HO-B04A, Yorktown Heights, NY 10598; (914) 789-7626. For electronic submission, Shriver's addresses are
Compmail +, b.shriver; CSnet, Shriver@ibm.com;
Vnet, shriver at yktvmh.

Staff

Editor and Publisher: True Seaborn
Managing Editor: Marilyn Potes
Issue Editor: Nancy Hays
Assistant Editors: Stephen Newland, Chuck Governale
Contributing Editor: Ware Myers
Assistant to the Publisher: Pat Paulsen
Art Director: Jay Simpson
Design/Production: Miriam Wiegel
Membership Manager: Christina Champion
Circulation Manager: Paul Zive
Advertising Director: Dawn Peck
Advertising Coordinators: Heidi Rex, Marian Tibayan

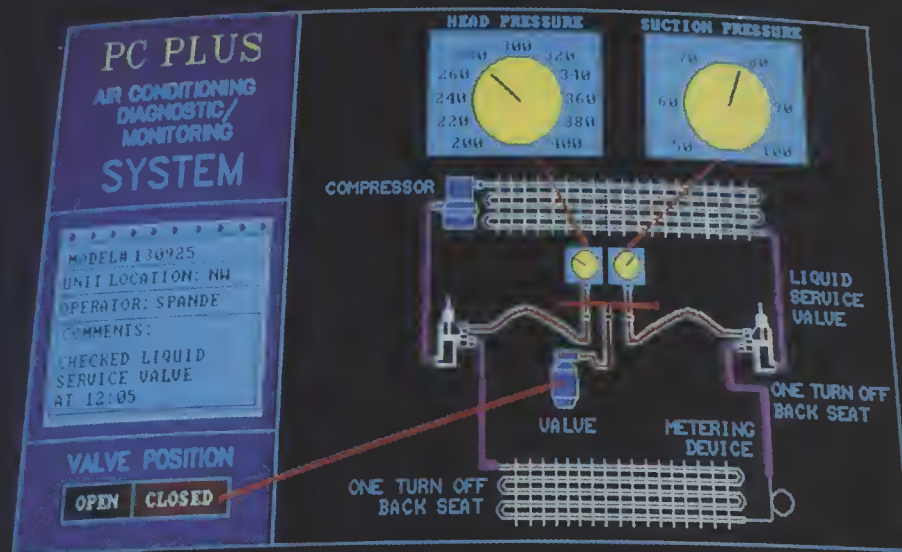
Magazine Advisory Committee

Michael Evangelist (chair), Vishwani D. Agrawal, James J. Farrell III, Ted Lewis, David Pessel, True Seaborn, Bruce D. Shriver, John Staudhammer

Publications Board

J. T. Cain (chair), Vishwani D. Agrawal, J. Richard Burke, Gerald L. Engel, Michael Evangelist, James J. Farrell III, Lansing Hatfield, Ronald G. Hoelzeman, Ted Lewis, Ming T. Liu, Ez Nahouraii, David Pessel, C.V. Ramamoorthy, Vincent Shen, Bruce D. Shriver, John Staudhammer, Steven L. Tanimoto

Texas Instruments has system developers need.



“Personal Consultant™ Plus...offers a very fine expert system development and delivery tool that already has a proven record with end-users.”

— Susan Shepard, *AI Expert*

- Personal Consultant Plus 3.0 Standard Features**
- Frames, rules, meta rules and procedures
 - Forward/backward chaining
 - Confidence factors
 - Regression testing and rule tracing
 - End-user explanation facilities
 - Graphics image capture and display
 - Interfaces to dBase™, Lotus 1-2-3™, DOS files, EXE or .COM programs, “C”
 - Complete LISP development environment
 - 2-megabyte expanded/extended memory support
 - Mouse support
 - Context sensitive help
 - “Getting Started” tutorial-style manual

- Personal Consultant Images**
- Optional add-on package to PC Plus (3.0)
 - Allows integration of “active images” into

what serious expert Power tools.

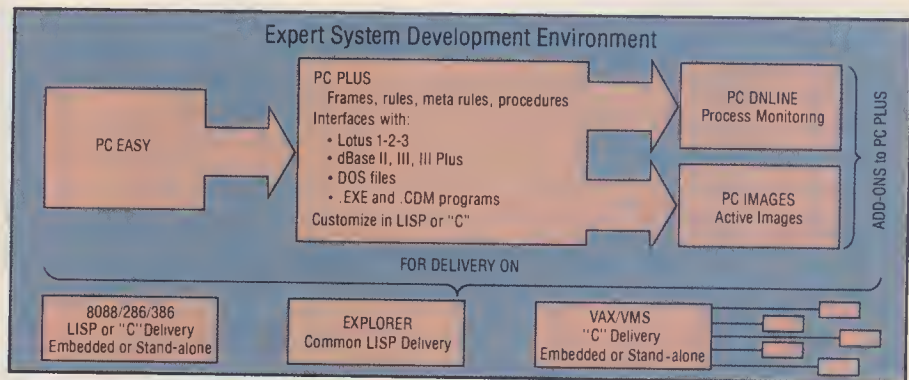
Among all the expert system development tools available for personal computers today, none deliver the power and flexibility of TI's Personal Consultant series.

Personal Consultant Easy is ideal for getting started, and is upwardly compatible with the higher functionality of PC Plus. For experienced developers, Personal Consultant Plus and its optional add-on enhancements, Online and Images, were designed to help solve a broader range of complex problems.

package helps deliver expertise that is "online all the time."

Application delivery as flexible as the tools themselves.

Delivery can be in LISP for flexibility, or "C" for maximum speed and portability. Our "C" options support either stand-alone or "embedded" knowledge bases. Options are available for DOS-based PCs, TI's Explorer, and DEC's VAX™ line of multi-user minis running under VMS™.



Personal Consultant Plus. Full power for an affordable price.

At \$2,950, PC Plus has proven to be one of the richest and most flexible problem-solving tools available for the development of complex knowledge-based systems. Designed to take advantage of today's more powerful 286/386 DOS-based computers, or TI's Explorer™ Symbolic Processing System, the new 3.0 version of PC Plus provides powerful standard features and a continuing growth path with the addition of either PC Images or PC Online, or both.

Personal Consultant Images. Picture an expert system with interactive graphics.

At \$495, PC Images enables developers to create knowledge-based applications that incorporate complex graphical "active images." User-interactive dials, gauges, forms and selection images provide a more exciting visual data input and output style.

Personal Consultant Online. The expert system as part of the process. At \$995, PC Online allows the developer to design expert systems which interact directly with process data, as opposed to input from a human operator. Designed for intelligent process monitoring applications, this optional

"Texas Instruments has done more than any other company to educate people about AI, to popularize it, and to make useful AI tools available at reasonable prices."

— Jim Seymour, *PC Magazine*.

Technical support, training courses and Knowledge Engineering Services are available for the Personal Consultant products. If you have a question about any of our expert system power tools, we have the answer.

Pick up the phone and gain a powerful advantage.

Call 1-800-527-3500 for technical overviews of our products and a PC Plus case histories brochure which details how our power tools are being put to work today.

36106

© 1987 TI

Personal Consultant and Explorer are trademarks of

Texas Instruments Incorporated.

dBase is a trademark of Ashton-Tate.

Lotus 1-2-3 is a trademark of Lotus Development Corp.

VAX and VMS are trademarks of Digital Equipment Corporation.

*Available 4Q 1987.

**TEXAS
INSTRUMENTS**

- knowledge bases
- Interactive dials, gauges, forms and selection images
- Multiple images can be combined on same screen
- "Getting Started" tutorial-style manual

Personal Consultant Online

- Optional add-on package for PC Plus (3.0)
- ONLINE expert systems that interact directly with process data
- Multiple interfaces to data acquisition and analysis programs
- Knowledge base synchronization with process data
- Functions for historical and predicted trends
- Special user interface/reporting capabilities
- "Getting Started" tutorial-style manual



Roy Russo

The Computer Society exists to serve its members, but the reasons for joining may not be obvious. In this message, Merlin Smith explains the benefits of belonging. (Incidentally, for a general list of member services, please refer to the Membership Information Page on the inside back cover.)

Roy Russo
President

The benefits of belonging

Look for the reward. When your message accompanies the two mug shots on this page, you had better promise a reward. We know that readers ranked officers' messages as *Computer's* least memorable feature in a poll some years ago, but we do have an important message. So please read on — there *is* a reward.

Why belong? We were asked that question recently, and there was a ghastly pause. Clearly the matter for a committee study, I reasoned, and another member poll (the typical professional reaction to a difficult issue). Well, we have conducted a member poll of why members join (and drop out), how they view our services, and what they want from our society. The society's services were perceived very well, and this strengthened our desire to make membership services even better in the future. Members cited several reasons for joining. Forty-four percent said our magazines were the main reason. Twenty-one percent stated that "belonging was part of being a professional." A surprisingly high 14 percent of members joined because of standards activities. The next 15 percent was divided about equally among "publishing papers," "member discounts for publications and conferences," and "keeping up with the field."

Actually, our polling reflects a continuing determination to have the society serve its members and the profession. We solicit your suggestions for improving services such as meetings, publications, tutorials, technical committees, and standards. And tell us about our problems. Every letter is read by at least two people, and our ombudsman program attempts to rectify every member complaint.

Our objective. Keeping members a step ahead in their profession is our objective. The society's programs are now at their peak performance, with probably the best overall leadership ever. We're excited about Bruce

Shriver's plans for *Computer* (see p.10), already rated very highly in the member poll. (This is the publication that all members receive.) You can look forward to lively and timely treatment of topics, a special series of articles by industry leaders, and new and rejuvenated features. According to Shriver (and others), "We gain respect the old fashioned way. We earn it."

The society's five other magazines provide a practical application-oriented treatment of the leading edge of computer technology, and the three transactions provide the more analytical and theoretical base — a unique combination of both theory and practice. And the society sponsors over 80 meetings annually, many the premier conferences in the industry. Your society is also a major publisher of nonperiodical computer-related materials — and there's more.

Yet, each year many members drop their memberships, both in the IEEE and the Computer Society. Members can vote with their pocketbooks, and that's not all bad. But what if nobody belonged? (An obvious benefit, you wouldn't be reading this message.) Shouldn't everybody belong — a "part of being a professional"? A dying tradition you say, no longer a tenet of our educational system? Still, if everybody belonged, we'd have all the good people. It's vitality, not just numbers, that counts. We're an organization driven by volunteers. If more people belonged, there would be more volunteers, more programs, more services for members. That's us.

Our greatest challenge. The commitment of students to life-long participation in the profession is our greatest challenge. Students receive society services at major discounts. Too few join. Too few leave school with the sense that membership is important to their careers or to their employers. We have to work on this problem, and we solicit your help and ideas.

Our surveys indicate that many mem-



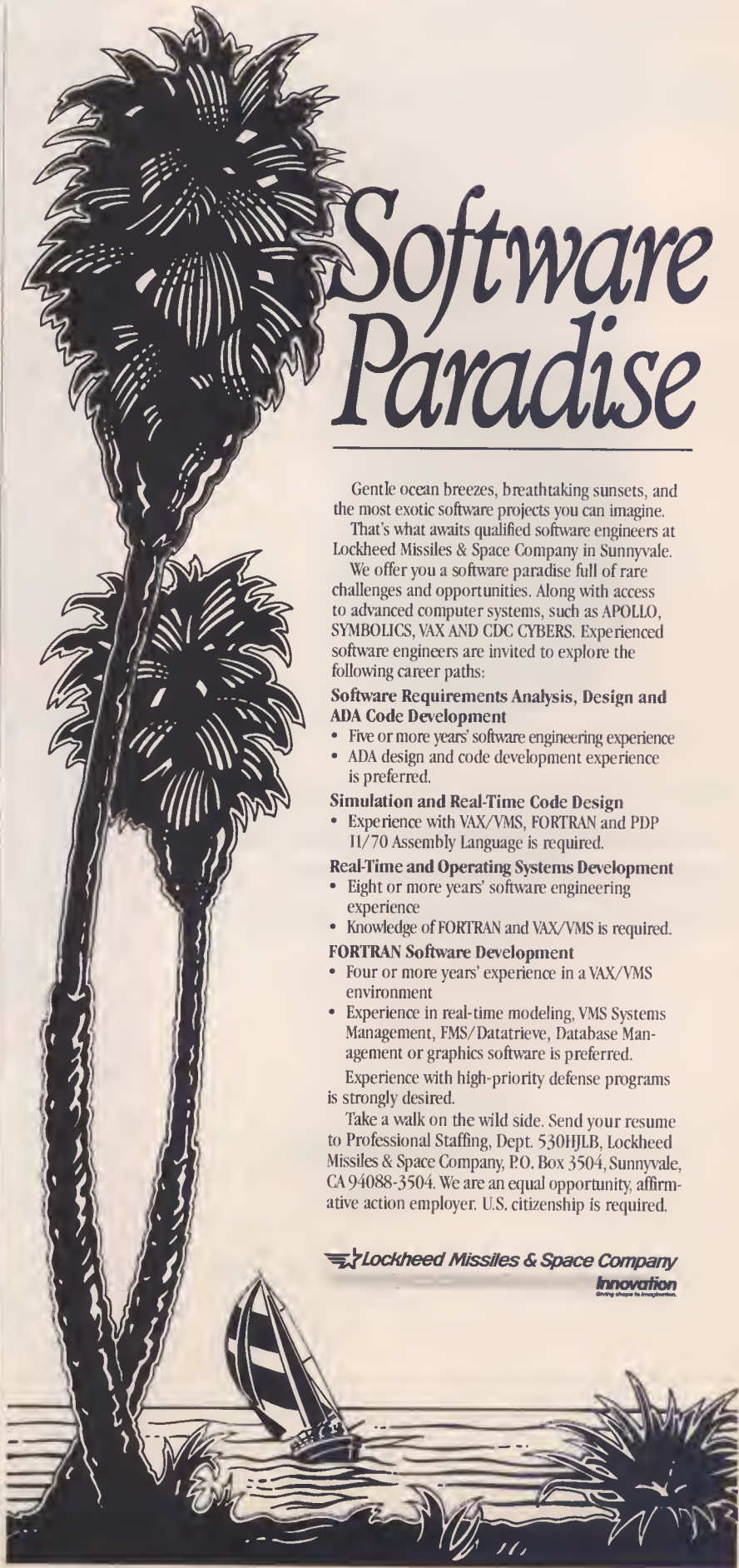
Merlin Smith

bers are unfamiliar with the society's services. This year, around dues renewal time, we will be generating and mailing a member handbook. Watch for it. We hope it will encourage your participation in more society activities and strengthen your resolve to renew your membership. Please show it to your nonmember colleagues as well. You can receive an additional copy of the handbook by circling number 202, Membership Information, on the Reader Service Card in this issue.

Now if you are still with me, you have to be the salt of the earth, or at least a good member, and you know others who would benefit by becoming members. We don't have their names and addresses. Send them to us and we'll write them. (Send to Merlin Smith, Membership, Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.)

And now the reward: you'll have our everlasting gratitude and — better still — a stronger profession. Lend us a hand.

Merlin Smith
Vice President, Membership and
Information



Software Paradise

Gentle ocean breezes, breathtaking sunsets, and the most exotic software projects you can imagine.

That's what awaits qualified software engineers at Lockheed Missiles & Space Company in Sunnyvale.

We offer you a software paradise full of rare challenges and opportunities. Along with access to advanced computer systems, such as APOLLO, SYMBOLICS, VAX AND CDC CYBERS. Experienced software engineers are invited to explore the following career paths:

Software Requirements Analysis, Design and ADA Code Development

- Five or more years' software engineering experience
- ADA design and code development experience is preferred.

Simulation and Real-Time Code Design

- Experience with VAX/VMS, FORTRAN and PDP 11/70 Assembly Language is required.

Real-Time and Operating Systems Development

- Eight or more years' software engineering experience
- Knowledge of FORTRAN and VAX/VMS is required.

FORTRAN Software Development

- Four or more years' experience in a VAX/VMS environment
- Experience in real-time modeling, VMS Systems Management, FMS/Datatrieve, Database Management or graphics software is preferred.

Experience with high-priority defense programs is strongly desired.

Take a walk on the wild side. Send your resume to Professional Staffing, Dept. 530HJLB, Lockheed Missiles & Space Company, P.O. Box 3504, Sunnyvale, CA 94088-3504. We are an equal opportunity, affirmative action employer. U.S. citizenship is required.

 **Lockheed Missiles & Space Company**
Innovation
driving change in imagination

Upgrade your knowledgeware.

Join the Computer Society today.



Where can you find, from one source, the most complete, current, reliable information on computer technology today?

**The Computer Society
of the IEEE.**

**Join us
and enjoy these benefits:**

- Automatic subscription to *COMPUTER Magazine*
- Discounts on other Computer Society and IEEE periodicals
- Discounts on Computer Society books
- Local chapter involvement
- Discounts on Computer Society conference registration
- Technical committees and standards task forces
- Electronic mail service at low member rates



Computer

- hardware and software designs
- tutorials
- survey
- product and industry news (monthly)

Included with membership

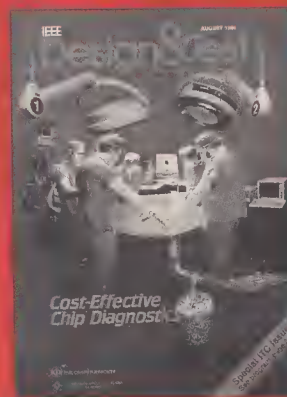


IEEE Computer Graphics and Applications

- displays
- computer-aided design and manufacturing
- computer animation
- solid modeling
- hardware software tools for rendering and graphics

(bimonthly)

Member rate: \$19/year



IEEE Design & Test of Computers

- design automation
- computer-aided design
- workstations and test equipment
- design for testability
- test strategies for memory and logic
- test of components, assemblies, and systems
- quality assurance
- VLSI

(bimonthly)

Member rate: \$20/year

IEEE Transactions on Computers

- theory, design, and practice of computer systems
- circuits, systems, and networks
- VLSI and digital devices
- fault-tolerant computing
- computational methods
- testing and performance evaluation
- computer applications

(monthly) **Member rate: \$18/year**

IEEE Transactions on Software Engineering

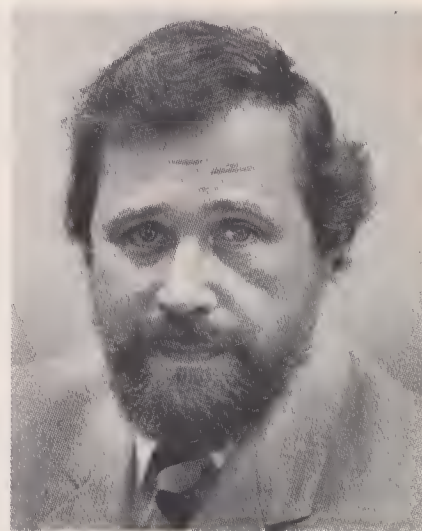
- specification, design, development, management, testing, maintenance, and documentation of software systems
- programming methodology
- programming environments
- software project management
- programming tools
- hardware and software monitoring

(monthly) **Member rate: \$18/year**

IEEE Transactions on Pattern Analysis and Machine Intelligence

- knowledge representation
- natural language processing
- inference systems
- group searching
- image processing
- statistical and structural pattern recognition
- robotics

(bimonthly) **Member rate: \$15/year**



Bruce D. Shriver

Changes—obvious and subtle

In my April 1987 editor-in-chief's message, I described several departments and features planned for *Computer*. Since then, I have been introducing the changes on a regular basis.

Sallie Sheppard, editor of the newly formed Computer Society News Department, brings readers timely and informative news items concerning the society's various activities. Ed Gallizzi has started a companion department, the Conferences Department, focusing on important technical highlights of society-sponsored conferences. Wiley McKenzie, as the new editor of the Book Reviews Department, has broadened its coverage not only in terms of the topics and the publishers represented, but also in the scope of media reviewed.

In last month's issue, we introduced the New Product Reviews Department headed by Dick Eckhouse. These reviews will provide both background to and representative samples of specific product classes.

This issue of *Computer* also contains new features.

Point/Counterpoint. Helen Wood, editor of the Standards Department, introduces the Point/Counterpoint format for discussing standards-related issues — a format that will be frequently used in the months ahead to

broaden the discussion in the increasingly important standards area.

Survey/Tutorial Series. I am also introducing our new Survey and Tutorial Series with several manuscripts in this issue. Surveys will be detailed reviews of the state of the art or the state of the practice in specific areas. Tutorials will be technically substantive introductions to given topics. Typically, both will contain substantial reading lists to aid readers interested in pursuing the topic in more depth.

I hope, as the pipeline becomes full, to have either a tutorial or survey in each issue of *Computer*. In particular, each theme issue will have a tutorial to provide background information for readers not working in the focus area, thereby helping them to understand and appreciate the articles devoted to the theme topic. I also hope to include more project surveys of the type that Guest Editors Jose Fortes and Ben Wah put together for their July 1987 special issue on systolic arrays. Such surveys should give readers a good idea of the variety of industrial, governmental, and academic efforts in a given area and will provide points of contact for these activities.

Reader Service Card. One of the more subtle changes in *Computer* has been our increased use of the reader service card to provide an easy commu-

nication path to our TCs, department editors, authors, and Computer Society services. For example, in this issue you can use the reader service card to

- obtain more extensive reports from several authors regarding the work described in their articles (see pp. 41, 57, and 69);
- tell us who you agree with in the Point/Counterpoint dialog in the Standards Department (p. 96);
- let us know how you feel about Section 1705 of the new tax law (p. 99);
- request detailed information on the Security and Privacy TC and the Microprocessors and Microcomputers TC, both of which are featured in the Computer Society News Department (pp. 100-101);
- obtain information on a new contest — computer bridge (p. 102);
- request a variety of membership and publication items (inside back cover).

While you're checking the card in response to one of the above items, don't forget to request additional information from that vendor whose ad you read. Our advertisers (who help pay our bills!) track results. Inquiries and sales leads, identifiable as originating from their ad in *Computer*, give a clear signal that their advertising dollars are being well spent.

Bruce D. Shriver
Editor-in-Chief

Now you can have perfect Fortran to C conversion ...at the touch of a button.

Here's your chance to take your entire Fortran library and convert it all to C.

Because FORTRIX™-C is here...the only software available that translates Fortran files to C code...automatically, perfectly and completely.

It doesn't matter which Fortran dialect, style or sequence you have...FORTRIX will convert it all. And because it's a machine translation, you get all the cost advantages of extraordinary speed. That's 600 lines a minute. And 50,000 lines that are ready to run in two weeks.

What's more, FORTRIX is proven. Right now it's on site and working worldwide at over 100 locations... including IBM,™ AT&T,™ Mitsubishi, U.S. Army and many others.

Find out how easy it is to keep your Fortran programs and still move up to C. Call our toll free number and we'll tell you exactly what FORTRIX will do for you. If it's more information you want, fill out this coupon and we'll send you technical details.

FORTRIX-C

The last word in Fortran to C conversion

Rapitech Systems Inc.
Montebello, Corporate Park
Suffern, NY 10901

(914) 368-3000 1-800-FORTRIX. Telex 509210

| Name _____
| _____
| Title _____
| _____
| Company _____
| _____
| Address _____
| _____
| City _____ State _____ Zip _____
| _____
| Phone _____

THE 1987 FALL JOINT COMPUTER CONFERENCE

INFOMART — DALLAS, TEXAS

October 25-29, 1987

EXPLORING TECHNOLOGY TODAY AND TOMORROW

Keynote Speaker — MAX D. HOPPER

Senior Vice President — Information Systems — American Airlines

The Fall Joint Computer Conference '87, a forum for ACM's and IEEE's Computer Society's Annual Conference, is to be held at Infomart in Dallas, Texas on October 25-29, 1987.

The Professional Education Program (PEP), scheduled for the first two days of this conference, will provide intensive one-day courses and hands-on workshops in five areas of technology and also new this year — Related Issues. Take a look at the schedule for Sunday and Monday, October 25 and 26, 1987.

The technical conference, which begins Tuesday, October 27, 1987 will afford an opportunity of listening to world reknown speakers from six different technologies and also a Related Issues Track.

In addition to the conference program and PEP seminars, the Exhibition Hall at INFOMART will be filled with exhibits containing the latest in hardware and software.

PEP Program Tracks

- Artificial Intelligence
- Software Systems
- Software Engineering
- User Engineering
- Modeling and Measurement
- Related Issues

Conference Program

- Software
- International
- Super Computers
- Algorithm Design/Security
- Database/Computer-Aided-Engineering
- Artificial Intelligence
- Related Issues

Exhibits — A Few Are

IBM	TIC Software
XEROX	EPSON
Intellicorp	NCR

THE FJCC '87 CONFERENCE PROGRAM

Special Evening Sessions — Wednesday, October 28, 1987

• Exploring the Limits of Uniprocessor Performance

Mr. Carl Amdahl, CGA Associates
Panel Chair with T. Aggerwala, IBM;
S. Chen, Cray Research;
R. Tomasulo, CGA Associates

• Computer Science Education in Latin America

Professor Paulo Franca, Federal University of
Rio de Janeiro, Panel Chair
with R. Monaco, University El Sur
National, Argentina;
O. Harafie, Organization of American States;
J.C. Amsalmi, UNESCO

• Heterogeneous Databases: Problems, Non Problems and Challenges

Dr. Ahmed Elmagarmid, The Pennsylvania
State University, Panel Chair
with A.P. Sheth, Honeywell Computer
Science Center;
J. Menon, IBM Almaden Research Center
C. Pu, Columbia University;
T.M. Ozsu, University of Alberta;
P. Valduriez, MCC;
R. Elmasri, University of Houston

• The Development of Expert Systems — Some Pragmatic Issues

Ms. Lorraine M. Duvall, Duvall Computer
Technologies, Inc., Panel Chair

**FALL JOINT
COMPUTER CONFERENCE
October 25-29, 1987
INFOMART
DALLAS, TEXAS**

**FJCC '87 CONFERENCE
REGISTRATION FORM**

Please send this registration form with payments to FJCC '87, INFOMART, 1950 Stemmons Freeway, #6038, Dallas, TX 75207. Phone: 1-800-322-FJCC. Advance registration discounts are available to all registration forms received, with full payment, postmarked on or before September 25. Attendees who have preregistered for the conference or the Professional Education Program may pick up their conference credentials beginning Saturday, October 24 from 3:00-5:00 pm at INFOMART on the plaza level. Please bring your registration acknowledgement with you. Notices of cancellation must be received in writing by INFOMART no later than October 15. There will be a \$25.00 administrative fee on all cancellations.

Name _____
(last) (first)

Position _____

Institution/Co. _____

Address _____

City _____ State _____ Zip _____

Phone _____

Mail List Restrictions:

- ☐ No Restrictions
☐ Professional Society Use Only

To qualify for member or student member discount.

☐ ACM _____

☐ IEEE-CS member number _____

NOTE:

Circle Fee for Registration Type Selected

FJCC '87 Conference	Advance Registration (Before 9-25-87)	Regular On-Site Registration
Full Conferences & Exhibits		
Member	\$190	\$225
Non-Member	\$240	\$285
Full-time Student	\$ 60	\$ 60
One Day Only		
* Member	\$ 75	\$100
* Non-Member	\$ 90	\$115
Circle One:		
Tu We Th		
* Exhibits Only	\$ 25	\$ 25
Proceedings Only	\$ 40	\$ 80
Professional Education Program Course		
One Day Courses		
Member	\$200	\$240
Non-Member	\$250	\$300
* No Proceedings		

A check or money order made payable in US funds to FJCC '87 for the full amount of the registration must accompany your registration form. Purchase orders will be accepted. The following credit card authorization may also be used:

U.S. \$ _____

Name as it appears on card: _____

Card No. _____

☐ VISA ☐ AMER. EXP. ☐ MC

Exp. Date: _____

Authorized Signature _____

Date _____

☐ Purchase Order

(Copy of Purchase Order must accompany registration)

☐ I plan to attend the Complimentary Special Joint Conference and Exhibitor's Reception on October 27, 1987.

☐ I plan to attend the complimentary INFOMART Tenant Reception & Tour on October 26, 1987.

☐ Spouse Program Information

FJCC HOTEL RESERVATION FORM

October 25-29, 1987

Loews Anatole

Travelodge
(formerly Viscount)

Single	\$87	\$59
Double	\$99	\$59

Please indicate room preference by circling the appropriate rate. 9% tax is applicable to all above rates.

Arrival Date _____ Departure Date _____

Name _____

Affiliation _____

Address _____

City _____ State _____ Zip _____

Accompanying Person(s) _____

Reservations must be received by appropriate hotel by October 4, 1987.

Deposits are required if you plan on arriving after 6 p.m. to guarantee availability of your room. Deposits will be refunded by the hotel if you cancel at least 48 hours prior to the arrival date.

Loews Anatole Reservations — please send completed form to:

Loews Anatole Hotel — Headquarters Hotel
Reservations Department
2201 Stemmons Freeway
Dallas, TX 75207

Phone reservations will also be accepted by calling (214) 748-1200 or toll-free at (800) 223-0888.

Travelodge Reservations — please send completed form to:

DMC Services
FJCC 87
P.O. Box 58755
Dallas, TX 75207

Phone reservations will also be accepted by calling (800) 972-1163.

TRAVEL SERVICES

DMC Services has been named the official travel coordinator for FJCC '87. Special air fares have been arranged with American Airlines. A 40% discount off the cost of the coach fare in effect the date the tickets are purchased is available to FJCC attendees. Additionally, American Airlines is offering a 5% discount off the lowest applicable round trip fare, subject to availability.

Special car rental rates have also been arranged by DMC Services through Dollar Rent-A-Car. These rates start as low as \$29.00 a day or \$140 a week for an economy size car.

For information and/or reservations call toll-free (800) 972-1163.

Also — To be named during this joint conference of the ACM and IEEE is the Turing Award Recipient.

Computer Chess Tournament:

**The ACM's 18th North American Computer Chess Championship
1st, 2nd, and Final Rounds on October 25, 26 and 27.**

FJCC 87 Professional Education Program

Sunday — October 25, 1987 (9 am - 5 pm)	
Artificial Intelligence	
345	Computer Vision John R. Kender, Columbia University
380	Computer Architectures for Knowledge Processing Ghassan Z. Qadah, Northwestern University
340	Machine Learning Michael Lebowitz, Columbia University
370	Knowledge Acquisition Techniques for Expert System Development Karen L. McGraw, Texas Instruments
375	Speech Recognition: From Isolated Digits to Natural Language Dictation Paul Bamberg, Dragon Systems
390	Introduction to Building Expert Systems John McGregor, Murray State University
Modeling and Measurement	
110	Discrete-Event Simulation and Modeling Osman Balci, Virginia Polytechnic Institute and State University
Computer Design	
940	Data Flow Computing: Models, Languages and Machines Jayantha and Susantha Herath, Keio University
910	Using Laser Optical Technologies Greg Baur and D.V. Pigford, Western Illinois University
920	Introduction to Fault-Tolerant Computing with Applications Bill D. Carroll, University of Texas; Victor P. Nelson, Auburn University
970	Concurrent Processing Architectures Phil Neches, Teradata Corporation
Software Systems	
210	The UNIX™ Operating System John H. Carson, George Washington University; SW Productivity Consortium
250	An Ada® Primer J.L. Silver, K.O. Rehmer, L. Rising, M. Temte, Purdue University at Ft. Wayne
280	Applications Development Technologies: Fourth Generation Languages Donald R. Chand, Bentley College; Sri Raghavan, Wang Institute
Software Engineering	
450	Program Design Language (PDL): What It Does; How You Use It Richard H. Smith, Whittaker 4C, Torrance, CA
460	Computer-Aided Software Engineering for Business Applications Raymond Yeh, SYSCORP International
490	Database Systems Design Julia Hodges, Mississippi State University
620	Software Engineering Management for Quality and Cost Control Tom Gilb, Consultant
630	New Paradigm for Software Development William W. Agresti, Computer Sciences Corporation
645	Statistical Approach to Software Engineering C.K. Cho, Consultant
User Engineering	
520	Principles and Guidelines in User Interface Design Deborah Mayhew, Deborah J. Mayhew & Associates
Related Topics	
820	End-User Training: Developing Materials/Programs in a Microcomputer Environment Susan Helms, Hardin-Simmons University
840	Project Planning & Management: State-of-the-Art Strategies Lois Zells, Lois Zells & Associates
860	Dynamics of Expert Interviewing Michael Burson, Texas Instruments

Monday — October 26, 1987 (8 am - 4 pm)	
Artificial Intelligence	
385	Computer Architectures for Knowledge Processing Ghassan Z. Qadah, Northwestern University
350	Natural Language Processing Michael Lebowitz, Columbia University
320	LISP and AI Programming Harlan H. Black, US Army Communications/Electronics Command
330	Knowledge Based Expert Systems: Mergers of AI & DBMS Techniques Lois Boggess and Julia Hodges, Mississippi State University
360	Knowledge Acquisition: Gathering and Interpreting Expertise Roy Maxion, Carnegie Mellon University; Mark Detweiler, University of Pittsburgh
Modeling and Measurement	
115	Discrete-Event Simulation and Modeling Osman Balci, Virginia Polytechnic Institute and State University
Computer Design	
945	Data Flow Computing: Models, Languages and Machines Jayantha and Susantha Herath, Keio University
930	Design and Test of Reliable Fault-Tolerant Systems Dhiraj K. Pradhan and Adit Singh, University of Massachusetts at Amherst
950	Parallel Processing Networks and Systems Howard J. Siegel, Purdue University
960	VLSI Computer Architecture for GaAs Technology V. Milutinovic, Purdue University
Software Systems	
240	Hand-On C Programming Vess Lee Johnson, Mississippi State University
230	Software Development With UNIX™ John H. Carson, George Washington University; SW Productivity Consortium
260	Ada™ Tasking and Tasking Paradigms Edward Colbert, Absolute Software Company
290	Fault-Tolerant Distributed Software Ahmed Elmagam, The Pennsylvania State University
Software Engineering	
430	The Logical Basis for Computer Programming Zohar Manna, Stanford University; Richard Waldinger, SRI International
440	Object-Oriented System Design and Implementation John McGregor, Murray State University
475	Building Defect Free Software Michael E. Fagan, IBM TJ Watson Research
625	Software Engineering Management for Quality and Cost Control Tom Gilb, Consultant
480	Integrating Heterogeneous Distributed Databases: Requirements, Concepts and Solution Amit P. Sheth, Honeywell CSDD
610	Automated Configuration Management of the Software Life Cycle Kevin R. Nix, Softool Corporation
User Engineering	
535	User Engineering and Active Prototyping L. McLaughlin, L. Valt, S. Overmyer, TRW Defense Systems Group
Related Topics	
810	The Micro Information Center: Serving the End-User Christine Dobson and Robert M. Peden, INFOMART
830	Effective Management Communications David M. Rappaport, Arthur Andersen & Co.; Karen Quinn, University of Chicago
850	Financial Aspect of New Technology Enterprises R.G. Garrick and S.A. Myers, Ernst & Whinney

FJCC Presents the best in Professional Education Programs (PEP) with speakers who are leaders in their field of expertise. All courses are either hands-on workshops or lecture-style presentations and are one-day sessions. Sunday, October 25 or Monday, October 26, 1987. Due to space limitation, pre-registration for these programs is recommended. Additional information on PEPs is available from INFOMART, 1950 Stemmons #6038, Dallas, TX 75207 or call 214-746-3500 or 800-322-FJCC.

Exhibitors Forum/Technical Presentations

- Technical Architecture and Work Bench**
Deon Fair — Arthur Andersen & Company
- Supporting End User Computing**
Tom Samson — Arthur Young & Company
- Doing Your Own Technical Publishing**
Marta Volbrecht — CPT Corporation
- Harnessing the Power of Your Existing Computer Resources**
Chuck Barnes — CYB Systems, Inc.
- Networking PC's in A True Peer-to-Peer Environment Using Corvus PC-NOS**
Linda Baker — Corvus Systems, Inc.
- Running Data Processing Like A Business**
Gary W. Kirkham — Forecasting Planning Associates
- IDA — A Step Toward Integration of AI and DB Technologies**
Dr. Gabriel Jacobson — GTE
- Benefits of Lap-Top Applications at Northern Telecom**
Darrall Jennings — Grid Systems
- IBM Presents the New Personal System/2**
Marvin Bond — IBM
- UNIX Supermicrocomputer Systems: Today's Technology and Tomorrow's Forecast — NCR**
- Local Area Networks: The Communications Link**
Peggy Burt — Novell, Inc.
- Turn Your Lemmons Into Lemonade**
Jack D. Spencer — R's Data Service
- Texas Innovation Information Network System**
John Rodman — Lehr McKenzie
- UNISYS Position Statement and Product Overview of Knowledge Based Systems**
Les Baker — UNISYS
- Demonstration: 2 Newest members of Product Family**
Wang Information Services Corporation
- Nothing Much Happens Until A Document Changes Hands**
Joanna Mathieson — Xerox Corporation

Technical Sessions

SOFTWARE TRACK:

- **Alternative Software Development Paradigms**
Chaired by Dr. William Lively, Texas A&M University
- **Software Design Synthesis**
Dr. Winston W. Royce, Lockheed Software Technology Center — Chair
- **Software Design Synthesis for FAA**
Dr. Philip Yoh, FAA — Chair
- **Software Design Synthesis for Strategic Computing**
Dr. Carl Davis, University of Alabama — Chair
- **Can Software Quality Be Measured?**
John D. Musa, AT&T Bell Laboratories — Chair
- **Software Problems of Large Organizations**
John Manley, Computing Technology Transition, Inc. — Chair
- **Solution to Large Software Development Problems**
William E. Riddle, Software Productivity Consortium — Chair

INTERNATIONAL TRACK:

- **Advanced Workstations in Japan**
Dr. Jumihiko Kamijo, Information Technology Promotion Agency, Japan & Les Beladiz, MCC, Panel Co-Chairs
- **TRON and Data Flow Machines**
Prof. Ken Sakamura, The University of Tokyo, Japan — Chair
- **Computer Technology in Europe**
Dr. Herbert Weber, Universitate Dortmund, Germany — Chair
- **Computer Technology in Asia & Brazil**
Dr. Pete A. Ng, New Jersey Institute of Technology — Chair
- **Issues on Software Development in Japan**
N. Akira Onoma, Hitachi, SK, Japan — Chair
- **Technology War**
Dr. Michael A. Harrison, University of California at Berkeley — Chair
- **Future of Information Industry**
H.L. Poppel, Broadview Associates
- **Visual Languages**
Dr. Shi-Kuo Chang, University of Pittsburgh; Dr. Tadao Ichikawa, Hiroshima University, Japan — Co-Chairs

SUPER COMPUTER/PARALLEL COMPUTATION TRACK:

- **Super Computers/Parallel Computation**
Dr. Aloysius K. Mok, The University of Texas at Austin — Session Chair
- **Parallel Computation**
Dr. Andre van Tilborg, Office of Naval Research — Panel Chair
- **Algorithm Design for Parallel Architectures**
Dr. David Matula, Southern Methodist University — Chair

ALGORITHM DESIGN/SECURITY TRACK:

- **Algorithm Design**
Prof. Peter Kornerup, Aarhus University, Denmark — Session Chair
- **Computer Security**
Dr. Virgil D. Gligor, Virgil D. Gligor, Inc. — Session Chair

DATABASE/COMPUTER AIDED ENGINEERING TRACK:

- **Database: Design and Modeling**
Dr. David Hsiao, Naval Postgraduate School — Session Chair
- **Computer Aided Database Engineering**
Dr. Nick Roussopoulos, University of Maryland — Chair
- **Computer-Aided-Design**
Dr. Herbert Freeman, Rutgers University — Session Chair
- **Fault Tolerance**
Charles Graff, Center For Communications Systems — Session Chair
- **Communications**
Dr. Edwin C. Foudrat, NASA, Langley Research Center — Session Chair

ARTIFICIAL INTELLIGENCE TRACK:

- **Logic Programming & Architectures**
Prof. Masayuki Ida, Aoyama Gakuin University; Dr. Du Ghang, University of Illinois at Chicago — Session Chairs
- **Image Recognition/Signal Processing**
Dr. J.K. Aggarwal, The University of Texas at Austin — Chair
- **AI Theory**
Prof. P. Bruce Berra, Syracuse University — Chair
- **AI Application**
- **AI and Software Engineering**
Dr. David Yun, Southern Methodist University — Panel Chair

RELATED ISSUES TRACK:

- **Formal Networks: Information, Research, Capital**
John Rodman, Texas Innovation Information Systems — Chair
- **Technology Commercialization**
Meg Wilson, The University of Texas Center for Technology Development and Transfer — Chair
- **Major Legal Issues Caused by Recent Advances In Computer Technology**
Jerry Keys of Brown, Maroney, Rose, Barber & Dye — Chair
- **Finance Issues In Technology**
Ronald G. Garrick, Ernst and Whinney — Chairman
- **High Technology Marketing**
Dr. Raymond W. Smilor, IC² Institute — Chairman
- **Science, Math and Computer Education**
Dr. Cathy Peavler and Dr. Barbara ten Brink, Texas Education Agency — Co-Chairs
- **Technical Training and Continuing Education**
Dr. Thomas Stauffer, University of Houston at Clearlake — Chair
- **Computer Education: Today & Tomorrow**

For exhibitor information
contact
Cynthia Cegelski
Telephone (214) 746-3547
or
(800) 322-FJCC

Gould solutions adapt to your needs.

Gould has the flexibility to adapt to your situation with the skill of a chameleon.

Gould's 32-bit CONCEPT/32® series provides a lot more computer for the price. Using Gould's real-time operating system, MPX-32®, the CONCEPT/32 product line gives you the performance needed to handle your particular requirements.

The PowerNode® series gives you the flexibility and versatility of UNIX®: Gould's UTX/32 operating system is a unique combination of Berkeley BSD 4.2 with AT&T System V.

Our "Compatibility Suite" of application software packages are operational across the entire Gould

PowerNode series. . . the widest range of Unix-based systems in the world.

To succeed, Gould has evolved to meet the needs of our customers. Year after year. Gould continues to innovate with products like Hypersearch — a superior text retrieval system, and our unmatched, complete computer-based training package.

For more information on how we can adapt to your needs, contact Gould Inc., Information Systems Computer Systems Division, 6901 West Sunrise Boulevard, Fort Lauderdale, Florida 33313 1-800-327-9716.

High Performance Solutions in Factory Automation, Computers, Instrumentation, Defense, and Semiconductors.

CONCEPT/32, MPX-32, PowerNode and UTX/32 are registered trademarks of Gould Inc.
UNIX is a registered trademark of AT&T Bell Labs
Reader Service Number 4



GOULD
Electronics



Hypertext: An Introduction and Survey

Jeff Conklin

Microelectronics and Computer Technology Corp.

Most modern computer systems share a foundation which is built of directories containing files. The files consist of text which is composed of characters. The text that is stored within this hierarchy is linear. For much of our current way of doing business, this linear organization is sufficient. However, for more and more applications, a linear organization is not adequate. For example, the documentation of a computer program* is usually either squeezed into the margins of the program, in which case it is generally too terse to be useful, or it is interleaved with the text of the program, a practice which breaks up the flow of both program and documentation.

As workstations grow cheaper, more powerful, and more available, new possibilities emerge for extending the traditional notion of "flat" text files by allowing more complex organizations of the material. Mechanisms are being devised which allow direct machine-supported references from one textual chunk to another; new interfaces provide the user with the ability to interact directly with these chunks and to establish new relationships between them. These extensions of the traditional text fall under the general category of *hypertext* (also known as *nonlinear text*). Ted Nelson, one of the

**Hypertext systems
feature machine-
supported links—both
within and between
documents—that open
exciting new
possibilities for using
the computer as a
communication and
thinking tool.**

pioneers of hypertext, once defined it as "a combination of natural language text with the computer's capacity for interactive branching, or dynamic display . . . of a nonlinear text . . . which cannot be printed conveniently on a conventional page."¹

This article is a survey of existing hypertext systems, their applications, and their design. It is both an introduction to the world of hypertext and, at a deeper cut, a survey of some of the most important

design issues that go into fashioning a hypertext environment.

The concept of hypertext is quite simple: Windows on the screen are associated with objects in a database, and links are provided between these objects, both graphically (as labelled tokens) and in the database (as pointers). (See Figure 1.)

But this simple idea is creating much excitement. Several universities have created laboratories for research on hypertext, many articles have been written about the concept just within the last year, and the Smithsonian Institute has created a demonstration laboratory to develop and display hypertext technologies. What is all the fuss about? Why are some people willing to make extravagant claims for hypertext, calling it "idea processing" and "the basis for global scientific literature"?

In this article I will attempt to get at the essence of hypertext. I will discuss its advantages and disadvantages. I will show that this new technology opens some very exciting possibilities, particularly for new uses of the computer as a communication and thinking tool. However, the reader who has not used hypertext should expect that at best he will gain a perception of hypertext as a collection of interesting features. Just as a description of electronic spreadsheets will not get across the real elegance of that tool, this article can only hint at the potentials of hypertext. In fact, one must work in current hypertext environments for a while for the collection of fea-

*Documentation is the unexecutable English text which explains the logic of the program which it accompanies.

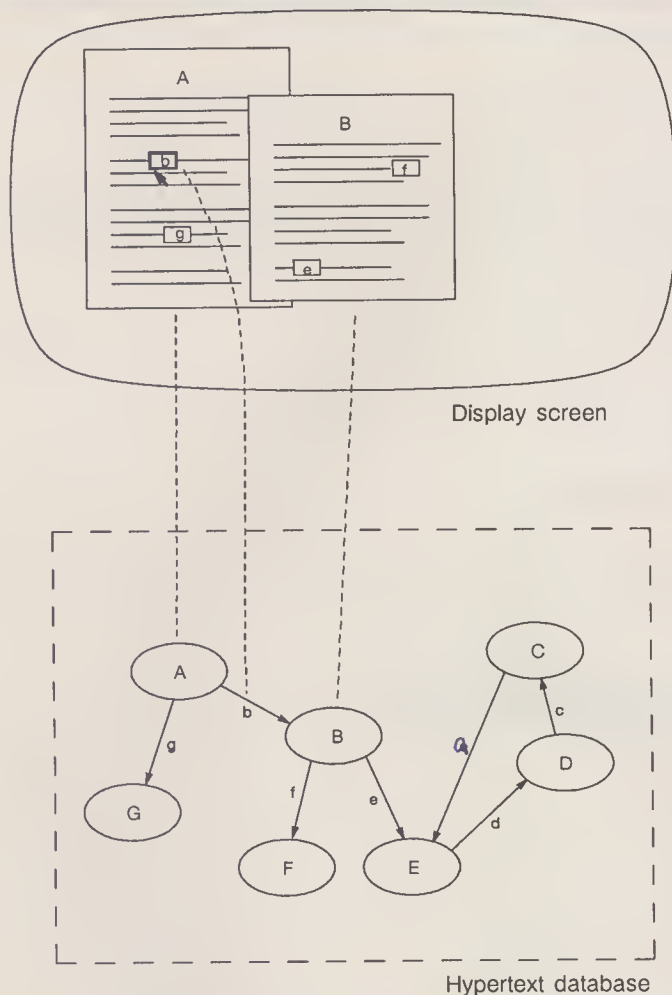


Figure 1. The correspondence between windows and links in the display, and nodes and links in the database. In this example, each node in the hypertext database is displayed in a separate window on the screen when requested. The link named "b" in window A has been activated by a pointing device, causing a new window named "B" to be created on the screen and filled with the text from node B in the database. (Generally, links can have names that are different from the name of the node they point to.)

tures to coalesce into a useful tool.

One problem with identifying the essential aspects of hypertext is that the term "hypertext" has been used quite loosely in the past 20 years for many different collections of features. Such tools as window systems, electronic mail, and teleconferencing share features with hypertext. This article focuses on machine-supported *links* (both within and between docu-

ments) as the essential feature of hypertext systems and treats other aspects as extensions of this basic concept.* It is this linking capability which allows a nonlinear organization of text. An additional feature

*While this article seeks to establish the criterion of machine-supported links as the primary criterion of hypertext, this is by no means an accepted definition. Therefore I will also review and discuss some systems which have a weaker notion of links.

that is common to many hypertext systems is the heavy use of windows that have a one-to-one correspondence with nodes in the database. I consider this feature to be of secondary importance.

One way to delimit hypertext is to point out what it is not. Briefly, several systems have some of the attributes of hypertext but do not qualify. Window systems fall into this category; while window systems do have some of the interface functionality, and therefore some of the "feel" of hypertext, window systems have no single underlying database, and therefore lack the database aspect of hypertext. File systems also do not qualify as hypertext; one could claim that a file system is a database, and that one moves among *nodes* (files) by simply invoking an editor with their names. However, to qualify as hypertext, a system must use a more sophisticated notion of links and must provide more machine support for its links than merely typing file names after a text editor prompt. Similarly, most outline processors (such as ThinkTank) do not qualify. They provide little or no support for references between outline entries, although their integrated hierarchical database and interface do approximate hypertext better than the other systems that I have mentioned. Text formatting systems (such as Troff and Scribe) do not qualify. They allow a tree of text fragments in separate files to be gathered into one large document; however, this structure is hierarchical and provides no interface for on-line navigation within the (essentially linear) document. Similarly, database management systems (DBMSs) have links of various kinds (for example, relational and object-oriented links), but lack the single coherent interface to the database which is the hallmark of hypertext.

As videodisc technology comes of age, there is growing interest in the extension of hypertext to the more general concept of hypermedia, in which the elements which are networked together can be text, graphics, digitized speech, audio recordings, pictures, animation, film clips, and presumably tastes, odors, and tactile sensations. At this point, little has been done to explore the design and engineering issues of these additional modalities, although many of the high-level design issues are likely to be shared with hypertext. Therefore, this survey will primarily address the more conservative text-based systems.

A glimpse of using hypertext. It is use-

ful to have a sense of the central aspects of using a hypertext system, particularly if you have never seen one. Below is a list of the features of a somewhat idealized hypertext system. Some existing systems have more features than these, and some have fewer or different ones.

- The database is a network of textual (and perhaps graphical) nodes which can be thought of as a kind of hyperdocument.

- Windows on the screen correspond to nodes in the database on a one-to-one basis, and each has a name or title which is always displayed in the window. However, only a small number of nodes are ever "open" (as windows) on the screen at the same time.

- Standard window system operations are supported: Windows can be repositioned, resized, closed, and put aside as small window icons. The position and size of a window or icon (and perhaps also its color and shape) are cues to remembering the contents of the window. Closing a window causes the window to disappear after any changes that have been made are saved to the database node. Clicking with the mouse on the icon of a closed window causes the window to open instantly.

- Windows can contain any number of link icons* which represent pointers to other nodes in the database. The link icon contains a short textual field which suggests the contents of the node it points to. Clicking on a link icon with the mouse causes the system to find the referenced node and to immediately open a new window for it on the screen.

- The user can easily create new nodes and new links to new nodes (for annotation, comment, elaboration, etc.) or to existing nodes (for establishing new connections).

- The database can be browsed in three ways: (1) by following links and opening windows successively to examine their contents, (2) by searching the network (or part of it) for some string,** keyword, or attribute value, and (3) by navigating around the hyperdocument using a browser that displays the network graphically. The user can select whether the nodes and links display their labels or not.

The browser is an important component

*Note that I am describing two uses of icons: those that function as placeholders for windows that have been temporarily put aside, and those within windows that represent links to other nodes.

**A string is a series of alphabetic and numeric characters of any length, for example "listening" or "G00274."

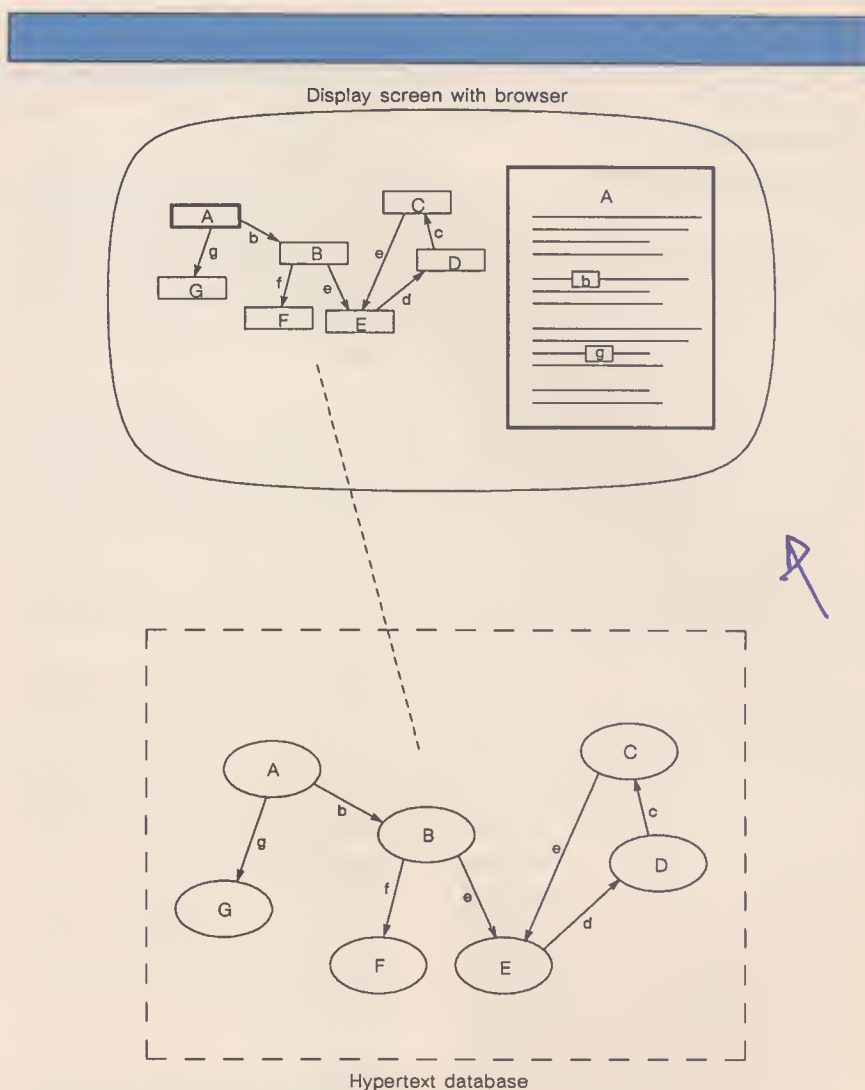


Figure 2. The screen at the top illustrates how a hypertext browser provides a direct two-dimensional graphic view of the underlying database. In this illustration, the node "A" has been selected for full display of its contents. Notice that in the browser view you can tell not only which nodes are linked to A but also how the subnetwork fits into the larger hyperdocument. (Of course, hyperdocuments of any size cannot be shown all at once in a browser—only portions can be displayed.)

of hypertext systems. As the hyperdocument grows more complex, it becomes distressingly easy for a user to become lost or disoriented. A browser displays some or all of the hyperdocument as a graph, providing an important measure of contextual and spatial cues to supplement the user's model of which nodes he is viewing and how they are related to each other and their neighbors in the graph. (See Figure 2.)

Using a browser can be likened to using visual and tactile cues when looking for a certain page in a book. Sometimes we remember the general way the page looked and about how far it was through the book, although we don't recall the page number or even which keyword terms would help us find it by using the index or table of contents. The browser display can be similarly scanned and scrolled when the

user has forgotten all but the appearance or location of a node.

Hypertext implementations

The history of hypertext is rich and varied because hypertext is not so much a new idea as an evolving conception of the possible applications of the computer. Many people have contributed to the idea, and each of them seems to have had something different in mind. In this section, I will review these theorists and their ideas in an effort to present a historical perspective as well as to sketch some of the hypertext applications that have been devised to date. I do not describe the individual systems and ideas reviewed here in any detail. For more detailed information, the reader is invited to consult the literature directly.

One kind of manual hypertext is the traditional use of 3 × 5 index cards for note taking. Note cards are often referenced to each other, as well as arranged hierarchically (for example, in a shoebox or in rubber-banded bundles). A particular advantage of note cards is that their small size modularizes the notes into small chunks. The user can easily reorganize a set of cards when new information suggests a restructuring of the notes. Of course, a problem with note cards is that the user can have difficulty finding a specific card if he has many of them.

Another kind of manual hypertext is the reference book, exemplified by the dictionary and the encyclopedia. In the sense that each of these can be viewed as a graph of textual nodes joined by referential links, they are very old forms of hypertext. As one reads an article or definition, explicit references to related items indicate where to get more information about those items. The majority of people's transactions with a dictionary make use of the linear (alphabetic) ordering of its elements (definitions) for accessing a desired element. An encyclopedia, on the other hand, can best be used to explore the local nodes in the "network," once one has found the desired entry through the alphabetic index.

There are also many documents in which references to other parts of the document, or to other documents, constitute a major portion of the work. Both the Talmud, with its heavy use of annotations and nested commentary, and Aristotle's writings, with their reliance on references to other sources, are ancient prototypes of

hypertextual representation.

But if one insists, as most modern proponents of hypertext do, that navigation through hypertextual space must be computer-supported in order to qualify as true hypertext, then the field is narrowed considerably, and the history likewise shortened.

In some ways, the people who first described hypertext—Bush, Engelbart, Nelson—all had the same vision for hypertext as a path to ultimate human-computer interaction, a vision which is still alive today among hypertext researchers. Thus the historical review below stresses the early development of ideas about hypertext as much as the more contemporary implementation efforts.

Because of the difficulty of precisely classifying hypertext systems according to their features, my description will list systems according to application. There are four broad application areas for which hypertext systems have been developed:

- macro literary systems: the study of technologies to support large on-line libraries in which interdocument links are machine-supported (that is, all publishing, reading, collaboration, and criticism takes place within the network);
- problem exploration tools: tools to support early unstructured thinking on a problem when many disconnected ideas come to mind (for example, during early authoring and outlining, problem solving, and programming and design);
- browsing systems: systems similar to macro literary systems, but smaller in scale (for teaching, reference, and public information, where ease of use is crucial);
- general hypertext technology: general purpose systems designed to allow experimentation with a range of hypertext applications (for reading, writing, collaboration, etc.)

These categories are somewhat informal. Often the single application to which a system has been applied to date determines which category it is described in. Bear in mind that some of the systems mentioned below are full-scale environments, while others are still only conceptual sketches. Some systems have focused more on the development of the *front end*

(the user interface aspects), while others have focused on the database issues of the *back end* (the database server). Table 1 identifies various features of the different hypertext systems which have been implemented.

Macro literary systems. The earliest visions of hypertext focus on the integration of colossal volumes of information to make them readily accessible via a simple and consistent interface. The whole network publishing system constitutes a dynamic corpus to be enriched by readers without defacing the original documents; thus, the difference between authors and readers is diminished. The advent of the computer has brought this vision closer to reality, but it has also revealed the monumental problems inherent in this application area.

Bush's Memex. Vannevar Bush, President Roosevelt's Science Advisor, is credited with first describing hypertext in his 1945 article "As We May Think,"² in which he calls for a major postwar effort to mechanize the scientific literature system. In the article, he introduces a machine for browsing and making notes in an extensive on-line text and graphics system. This *memex* contained a very large library as well as personal notes, photographs, and sketches. It had several screens and a facility for establishing a labelled link between any two points in the entire library. Although the article is remarkably foresightful, Bush did not anticipate the power of the digital computer; thus his *memex* uses microfilm and photocells to do its magic. But Bush did anticipate the information explosion and was motivated in developing his ideas by the need to support more natural forms of indexing and retrieval:

The human mind . . . operates by association. Man cannot hope fully to duplicate this mental process artificially, but he certainly ought to be able to learn from it. One cannot hope to equal the speed and flexibility with which the mind follows an associative trail, but it should be possible to beat the mind decisively in regard to the permanence and clarity of the items resurrected from storage.²

Bush described the essential feature of the *memex* as the ability to tie two items together. The mechanism is complex, but clever. The user has two documents that he wishes to join into a *trail* he is building, each document in its own viewer; he taps in the name of the link, and that name appears in a code space at the bottom of

Table 1. Hypertext systems and their features.

Hypertext Systems	Hierarchy	Graph-based	Link Types	Attributes	Paths	Versions	Procedural Attachment	Keyword or String Search	Text Editor	Concurrent Multi-users	Pictures or Graphics	Graphical Browser
Boxer	Yes	Yes	Fixed ¹	No ¹	No	No	Yes	Yes	Emacs	No	Yes	Yes
CREF	Yes	Yes	Yes	No	No	By link	No	Yes	Zmacs	No	Yes	No
Emacs INFO	Yes	No	No	No	No	No	No	Yes	Emacs	No	No	No
IBIS	Yes	Yes	Yes	No	No	By link	No	No	A basic text editor	Yes	No	No
Intermedia	Yes	Yes	Yes	Yes	No ²	No	No ²	Yes	Custom	Yes	Yes	Yes
KMS	Multiple	Yes	Fixed	No	No ¹	Yes	Yes	Yes	Text/graph. WYSIWYG	Yes	Yes	No
Neptune	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Smalltalk-80 editor	Yes	Yes	Yes
NLS/Augment	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Custom	Yes	Yes	No
NoteCards	Multiple	Yes	Yes	Nodes	No	No	Yes	Yes	Interlisp	Yes	Yes	Yes
Outline Processors	Yes	No	No	No	No	No	No	Yes	Various	No	No	No
PlaneText	Unix file sys.	Yes	No	No	No	No	No	Unix/grep	SunView text ed.	Yes	Yes	Yes
Symbolics Document Examiner	Yes	Yes	No	No	Yes	No	No	Yes	None	No	No	No
SYNVIEW	Yes	No	No	No	No	No	No	No	line ed./Unix	No	No	No
Textnet	Multiple	Yes	Yes	Yes	Yes	No	No	Keyword	Any	No	No	No
Hyperties	No	Yes	No	No	No	No	No	No ²	A basic text editor	No	Yes	No
WE	Yes	Yes	No	Fixed	No ²	No ²	No ²	No	Smalltalk-80 editor	No ²	Yes	Yes
Xanadu	No	Yes	Yes	Yes	Yes	Yes	No	No	Any	No	Yes	No
ZOG	Yes	No	No	No	No	No	Yes	Full text	Spec. Pur.	Yes	No	No

¹ Can be user programmed.

² Planned for next version.

In this table, each column represents one possible feature or ability that a hypertext system can provide. The negative or affirmative entries in the table indicate whether the corresponding hypertext system meets the standard criteria for a specified feature. These criteria are listed below.

Hierarchy: Is there specific support for hierarchical structures?

Graph-based: Does the system support nonhierarchical (cross-reference) links?

Link types: Can links have types?

Attributes: Can user-designated attribute/value pairs be associated with nodes or links?

Paths: Can many links be strung together into a single persistent object?

Versions: Can nodes or links have more than a single version?

Procedural attachment: Can arbitrary executable procedures be attached to events (such as mousing) at nodes or links?

String search: Can the hyperdocument be searched for strings (including keywords)?

Text editor: What editor is used to create and modify the contents of nodes?

Concurrent multiusers: Can several users edit the hyperdocument at the same time?

Pictures or graphics: Is some form of pictorial or graphical information supported in addition to text?

Graphics browser: Is there a browser which graphically presents the nodes and links in the hyperdocument?

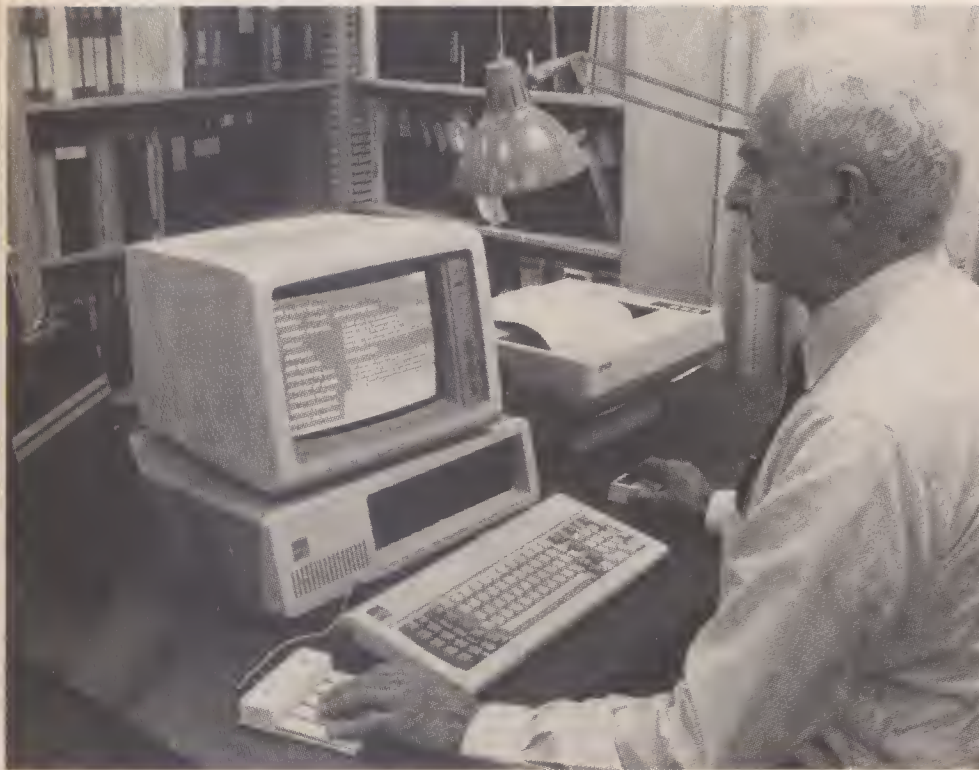


Figure 3. Engelbart at the NLS/Augment workstation. Note the chord key set under Engelbart's left hand. The chord key set is optional for Augment. It is a remarkable accelerator for character-driven commands and mouse-select screen operands.

each viewer; out of view, the code space is also filled with a photocell-readable dot code that names the other document and the current position in that document. Thereafter, when one of these items is in view, the other can be instantly recalled merely by tapping a button below the corresponding code space. Bush admitted that many technological breakthroughs would be needed to make his memex practical, but he felt that it was a technological achievement worthy of major expenditure.

Engelbart's NLS/Augment. Just less than two decades later Douglas Engelbart, at Stanford Research Institute, was influenced by Bush's ideas. In 1963, Engelbart wrote "A Conceptual Framework for the Augmentation of Man's Intellect."³ Engelbart envisioned that computers would usher in a new stage of human evolution, characterized by "automated external symbol manipulation":

In this stage, the symbols with which the human represents the concepts he is manipulating can be arranged before his eyes, moved, stored, recalled, operated upon according to extremely complex rules—all in very rapid response to a minimum amount of information supplied by the human, by means of special cooperative technological devices. In the limit of what we might now imagine, this could be a computer, with which individuals could communicate rapidly and easily, coupled to a three-dimensional color display with which extremely sophisticated images could be constructed . . .³

His proposed system, H-LAM/T (Human using Language, Artifacts, and Methodology, in which he is Trained), included the human user as an essential element: The user and the computer were dynamically changing components in a symbiosis which had the effect of "amplifying" the native intelligence of the user. This is still a common vision among developers of hypertext systems.

Five years later, in 1968, Engelbart's

ideas about augmentation had become more specific, and had been implemented as NLS (oN Line System) by the Augmented Human Intellect Research Center at SRI. NLS was designed as an experimental tool on which the research group developed a system that would be adequate to all of their work needs, by

placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our intercommunication, via the consoles.⁴

These consoles were very sophisticated by the standards of the day and included television images and a variety of input devices, including one of Engelbart's best known inventions, the mouse.*

Files in NLS were structured into a hierarchy of segments** called *statements*, each of which bore an identifier of its level within the file. For example, a document might have statements "1," "1a," "1a1," "1a2," "1b," etc., though these identifiers did not need to be displayed. Any number of reference links could be established between statements within files and between files. Note that this is a structure which is primarily hierarchical, but which allows nonhierarchical links as well. The importance of supporting both kinds of structures is a point to which I will return later. The system provided several ways to traverse the statements in files.

NLS, like other early hypertext systems, emphasized three aspects: a database of nonlinear text, *view filters* which selected information from this database, and *views* which structured the display of this information for the terminal. The availability of workstations with high resolution displays has shifted the emphasis to more graphical depictions of nodes, links, and networks, such as using one window for each node.

NLS provided viewing filters for the file structure: One could clip the level (depth) of hierarchy displayed, truncate the number of items displayed at any level, and write customized filters (in a "high-level content analysis language") that displayed only statements having the specified content. NLS also introduced the concept of

*Engelbart also introduced a five-key handset—a one-handed keyboard. The operator enters alphanumeric text by "chording" the five keys. Although this method is slower than two-handed typing, it has a considerable advantage for short commands when used with a mouse in the other hand.

**Segments were limited to 3000 characters in length.

multiperson distributed conferencing/editing.

NLS has evolved over the years. It is now called Augment (or NLS/Augment) and is marketed as a commercial network system by McDonnell Douglas. In developing NLS, the emphasis has been on creating a consistent environment for "knowledge workers" (that is, office automation for software engineers). The system now includes many forms of computer-supported communication, both asynchronous (email with links to all documents, journaling of ideas and exchanges, bulletin boards, etc.) and synchronous (several terminals sharing the same display, teleconferencing, etc.). It includes facilities for document production and control, organizational and project information management, and software engineering. (See Figures 3 and 4.)

Nelson's Xanadu Project. During Engelbart's development of Augment, another hypertext visionary, Ted Nelson, was developing his own ideas about augmentation, but with an emphasis on creating a unified literary environment on a global scale. Nelson coined the term "hypertext." His thinking and writing are the most extravagant of any of the early workers. He named his hypertext system Xanadu, after the "magic place of literary memory" in Samuel Taylor Coleridge's poem "Kubla Khan." In Xanadu, storage space is saved by the heavy use of links. Only the original document and the changes made to it are saved. The system easily reconstructs previous versions of documents. Nelson describes his objectives as follows:

Under guiding ideas which are not technical but literary, we are implementing a system for storage and retrieval of linked and windowing text. The *document*, our fundamental unit, can have windows to any other documents. The evolving corpus is continually expandable without fundamental change. New links and windows can continually add new access pathways to old material. Fast proprietary algorithms render the extreme data fragmentation tolerable in the planned back-end service facility.⁵

The long range goal of the Xanadu project has been facilitating the revolutionary process of placing the entire world's literary corpus on line. In fact, Xanadu's design makes a strong separation between the user interface and the database server, with most of the emphasis placed on the latter. In particular, great care has been taken that copyright protection remain-

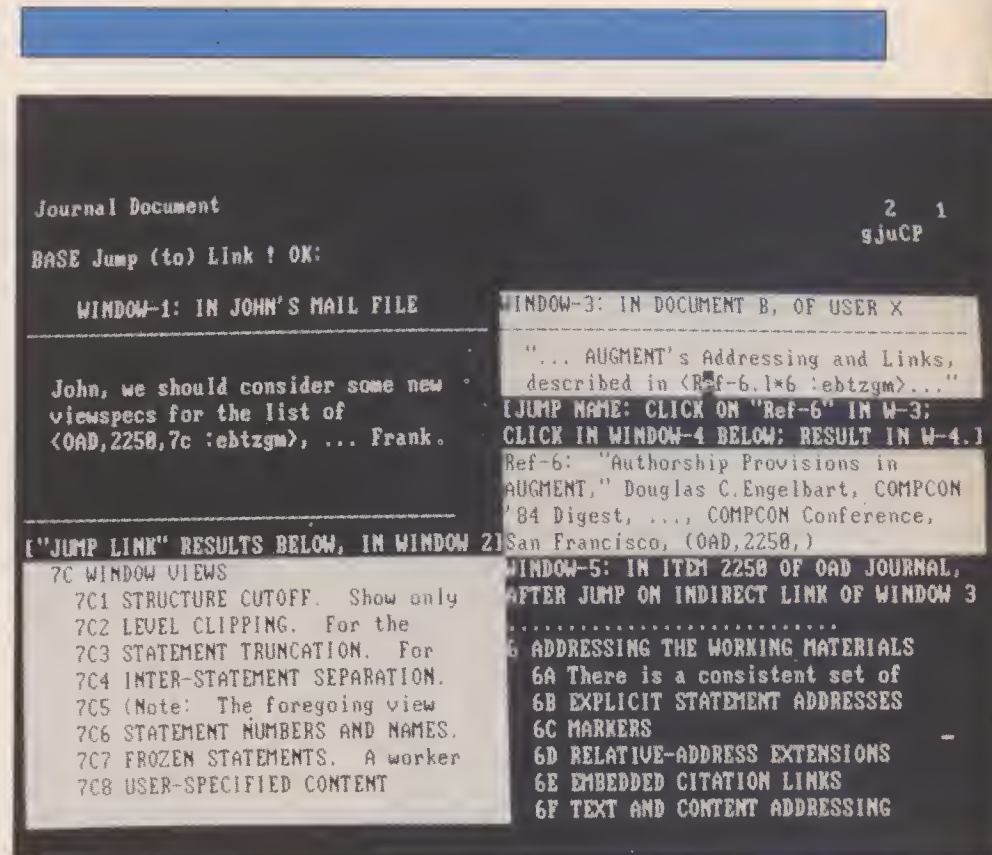


Figure 4. Augment display showing five windows. Window 1 (W-1) has a passage as if embedded in a message, showing a link to Branch 7c of Document 2250 in the OAD Journal. A ViewSpec ("ebtzgm") provides the following specifications: target level plus one, truncate to one line per statement, no blank lines between statements, show only that branch (e.g., not Branch 7d), and turn on Location Numbers. Window 2 (W-2) shows the view obtained with a jump link command. To perform a jump link command, the operator clicks on the link in W-1, then moves the cursor into W-2 for the final click. The very top-left system message announces that the desired Journal Item has been accessed, and the cluster at the top left of the screen verifies that the view is clipped to three levels and the statements truncated to one line each. Window 3 (W-3) shows an indirect link that specifies the linkage path. In effect, this link says "go to the statement in the file named 'Ref-6,' follow the link found there to its target file, and in that file find Location Number 6." Note that the same ViewSpec is specified here as for the link in W-1. Window 4 (W-4) identifies Ref-6 and provides its general reference source as the reference section at the end of the document; a user can jump from the link citation in W-3 to see this statement by using the jump name command. To perform this command, he clicks on "Ref-6" in W-3 then clicks on W-4. Window 5 (W-5) shows a view in the OAD-Journal Item 2250. The user can obtain this view by performing a jump link command on the indirect link of W-3. To perform this command, the user clicks on the indirect link of W-3 and then clicks in W-5.

tainable, and that a system for the electronic accounting and distribution of royalties is in place. Nelson predicts that the advent of on-line libraries will create a whole new market for the organization and indexing of this immense information store.

The back end of the Xanadu system has been implemented in Unix and is available in several forms, including as an on-line service (much like Engelbart's Augment). A crude front end for the Xanadu system is also available which runs on Sun workstations.

Trigg's Textnet. Randall Trigg wrote the first and to date the only PhD thesis on hypertext. In his thesis, he describes his Textnet system as supporting *nonlinear text*—text in which documents are organized as “primitive pieces of text connected with typed links to form a network similar in many ways to a semantic net.” The thesis focuses on specific link types that support literary criticism.

In the tradition of the field, Trigg's system is just a first step in the direction of his vision:

In our view, the logical and inevitable result [of the computer revolution] will be the transfer of all such [text handling] activities to the computer, transforming communication within the scientific community. All paper writing, critiquing, and refereeing will be performed on line. Rather than having to track down little-known proceedings, journals or unpublished technical reports from distant universities, users will find them stored in one large distributed computerized national paper network. New papers will be written using the network, often collaborated on by multiple authors, and submitted to on-line electronic journals.⁶

Textnet implements two basic types of nodes: those which have textual content (*chunks*) and those which hierarchically organize other nodes (tocs, for “table of contents”). Thus Textnet supports both hierarchical trees (via the toc nodes) and nonhierarchical graphs (via the typed links).

Trigg further proposes a specific taxonomy of link types for use by collaborators and critics in Textnet. He argues that there is generally a specific set of types of comments, and that there is a link type for each comment. For example, there are *refutation* and *support* links, and, more specifically, there are links to say that a point is irrelevant (“Pt-irrelevant”), that data cited is inadequate (“D-inadequate”), or that the style is rambling (“S-rambling”). Trigg describes over 80 such link types and argues that the disadvantage of having a limited set of link types is outweighed by the possibility of specialized processing on the hyperdocument afforded by a definite and fixed set of primitives.

In addition, Textnet supports the definition of *paths*—ordered lists of nodes used to browse linear concatenations of text and to dump such scans to hard copy. The path facility relieves the hypertext reader from having to make an *n-way* decision at each link; rather, the reader is provided a default pathway through the network (or part of the network), and can simply read the material in the suggested

order as if he were reading a linear document.

Trigg joined Xerox PARC after completing his thesis and was one of the principal architects of the Xerox NoteCards system.

Problem exploration systems. These are highly interactive systems which provide rapid response to a small collection of specialized commands for the manipulation of information. They can be thought of as the early prototypes of electronic spreadsheets for text and symbolic processing. One important feature of most of these tools is a facility for suppressing detail at various levels specified by the user. For example, the outline processors all have single keystroke commands for turning on and off the display of subsections of a section. This is an unusual but natural facility. Hypertext and similar tools excel at the collection of large amounts of relatively unstructured information. But such collections are of little use unless adequate mechanisms exist for filtering, organizing, and browsing. These are the primary desiderata of these authoring/thinking/programming systems.

Issue-Based Information Systems. Horst Rittel and his students have introduced the notion of Issue-Based Information Systems (IBIS)⁷ to handle systems analysis in the face of “wicked problems.”

Rittel describes wicked problems (as opposed to “tame” ones) as problems which cannot be solved by the traditional systems analysis approach (that is, (1) define the problem, (2) collect data, (3) analyze the data, (4) construct a solution). Wicked problems lack a definitive formulation; their problem space cannot be mapped out without understanding the solution elements; in short, the only way to really understand a wicked problem is to solve it. Wicked problems have no stopping rule. The design or planning activity stops for considerations that are external to the problem (for example, lack of time, money, or patience). Solutions to wicked problems are not “right” or “wrong”; they just have degrees of sufficiency. Rittel argues that solving wicked problems requires all those involved to exchange and argue their many viewpoints, ideas, values, and concerns. By coming to understand other viewpoints better, each par-

ticipant is able to understand the whole problem better. This process enables a common understanding of the major issues and their implications to emerge. IBIS is designed to support this design/planning conversation.

IBIS systems are thus a marriage of (1) teleconferencing systems which enable many people to participate in one conversation, and (2) hypertext, which allows participants to move easily between different issues and the different threads of argument on the same issue. The current version of Rittel's IBIS runs on an Apple PC and is being ported to Sun workstations.* IBIS has three types of nodes (*issues, positions, and arguments*), and uses nine types of relations to link these nodes. In a typical application, someone posts an issue; then that person or others post positions about that issue; and then the positions are argued using argument nodes. Of course, any of the three types of nodes can be the seed of a new issue. (See Figure 5.) The current set of relationships between nodes is: *responds-to, questions, supports, objects-to, specializes, generalizes, refers-to, and replaces.* The research on IBIS concentrates on ways to summarize and present the issue network, both for participants and decision makers.

Lowe's SYNVIEW. David Lowe's SYNVIEW system is similar in concept to Rittel's IBIS but goes in a different direction. It proposes that the participants, in addition to posting their own issues and arguments, assess previous postings as to their validity and relevance. The assessment is done by a kind of quantitative voting. For example, if you think that Joe's response to Sam makes a good point but is not really a direct response to Sam's posting, you might grade it “5, 1” (where 5 is a high validity rating and 1 is a low relevance rating). These values are averaged into the existing values for that posting. The various displays of the argument structure show the values for each posting, allowing readers to focus, if they choose to, on those argument trails having the highest voted validity.

Through debates on the accuracy of information and on aspects of the structures themselves, a large number of users can cooperatively rank all available items of information in terms of significance and relevance to each topic. Individual users can then choose the depth to which they wish to examine these structures for the purposes at

*A graphical Sun version, called gIBIS, is also being developed at the MCC/Software Technology Program.

hand. The function of this debate is not to arrive at specific conclusions, but rather to collect and order the best available evidence on each topic.⁸

UNC's WE. A group at the University of North Carolina at Chapel Hill has been developing a *writing environment* called WE.⁹ Their research is based on a cognitive model of the communication process which explains reading as the process of taking the *linear* stream of text, comprehending it by structuring the concepts hierarchically, and absorbing it into long-term memory as a *network*. Writing is seen as the reverse process: A loosely structured network of internal ideas and external sources is first organized into an appropriate hierarchy (an outline) which is then "encoded" into a linear stream of words, sentences, etc.

WE is designed to support the upstream part of writing. It contains two major view windows, one graphical and one hierarchical, and many specialized commands for moving and structuring material (nodes and links with attached text) between these two views. Normally a writer will begin by creating nodes in the graph view, where he can place them anywhere within the window. At this stage, little or no structure is imposed on the conceptual material. The writer can place nodes in "piles" if they seem to be related, or he can place individual nodes between two piles if they are somewhat related to both. As some conceptual structure begins to emerge from this process, the writer can copy nodes into the hierarchy window, which has specialized commands for tree operations. The hierarchy window has four different display modes: (1) the tree can be laid out on its side, with the root node on the left; (2) the tree can be hung vertically with the root at the top; (3) child nodes can be displayed inside their parent node; and, (4) the hierarchy can be shown in the traditional outline view.

WE uses a relational database for the storage of the nodes and links in the network. The user points with a mouse to select a node. A third window is an editor for the material within the currently selected node. A fourth window on the screen is for queries to the database. A fifth window is used to control system modes and the current working set of nodes.

WE is designed to be an experimental platform to study what tools and facilities will be useful in a writer's environment. The real validation of these ideas, as with so many of the systems described here, will

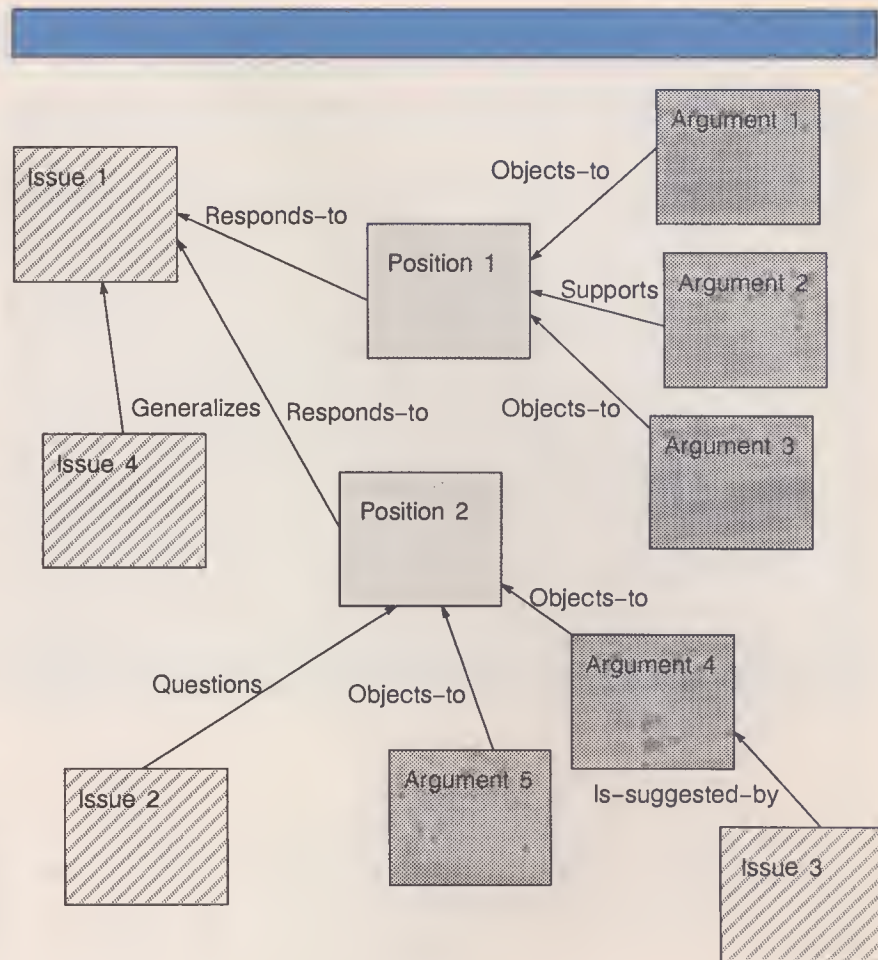


Figure 5. A segment of a possible IBIS-style discussion showing the topology of the IBIS network. Each node contains information on the type of the node, the time and date of creation, the author, a short phrase describing the content, a longer body of text with the text of the comment, a list of keywords, and a list of the incoming and outgoing links.

come with further experiments and analysis.

Outline processors. An *outline processor* is a word processing program which is specialized for processing outlines, in that its main commands deal with movement among, creation of, and modification of outline entries. In this respect, these programs commercialize many ideas from Engelbart's NLS/Augment. Outline processors also have at least simple text editors and do some text formatting, so that the user can use the same tool to go from outline to finished document. One of the most powerful features of outline processing is the ability to suppress lower levels of detail in the outline. As with Engelbart's NLS/Augment, the user can view just the top level of the outline, or the top *n* levels, or he can "walk the tree,"

opening up just those entries that are relevant or useful to the idea that he is working on. In addition, each outline entry can have a textual body of any length associated with it, and the user can make this body appear or disappear with a single keystroke. This feature is a real boon to the writing process, because it allows the user to have a view of both the immediate text that he is composing and the global context for it. It also facilitates rapid movement between sections, particularly in large documents, because in outline mode a remote section is never more than a few keystrokes away.

Most outline processors are personal computer programs, and they have done much to bring some of the concepts underlying hypertext into popularity. The first of these was called ThinkTank. It was released in 1984. It has since been joined

by a host of others, with names like MaxThink, Executive Writer/Executive Filer, Thor, Framework, Kamas, Fact Cruncher, Freestyle, Idea!, and PC-Outline.¹⁰ There are two very recent additions to the field: Houdini is an extension of MaxThink that supports rich nonhierarchical internode references; and For-Comment is a word processor that allows up to 15 people to apply hypertext-like annotations to a document (and can operate over a Local Area Network (LAN) in real time).

Aside from Houdini, most outline processors do not support inter-entry references, except by "cloning" the whole entry and displaying it in the new location. Only a few others provide windows for nodes. None of them provide explicit "mousable" link icons. For these reasons, one could argue whether they qualify as hypertext as I have defined it here. However, ThinkTank was the first program to be billed—somewhat pretentiously—as an "idea processor," and all of these programs treat sections of text as first-class objects and support manipulations that coincide with the way one manages ideas. They share these features with hypertext, and in this sense, they anticipate the inevitable proliferation of hypertext features within the mainstream of computer applications.

Structured browsing systems. The systems reviewed in this section were designed primarily for applications involving large amounts of existing information or requiring easy access to information. These systems pose different problems for their designers. Ease of learning and ease of use are paramount, and great care goes into crafting the interface. On the other hand, writing (adding new information) is usually either not allowed to the casual user or is not particularly well supported.

CMU's ZOG and Knowledge Systems' KMS. ZOG is a menu-based display system developed in 1972 at Carnegie-Mellon University.¹¹ It consists of a potentially large database of small (screen-sized) segments which are viewed one at a time. ZOG was developed with the particular goal of serving a large simultaneous user community, and thus was designed to operate on standard terminals on a large timesharing system. In 1981 two of the principals on the ZOG Project, Donald McCracken and Robert Akscyn, started the company Knowledge Systems and developed a commercial successor to ZOG

called Knowledge Management System (KMS).

Each segment of the ZOG/KMS database is called a *frame*. A frame has, by convention, a one-line title at the top of the screen, a few lines of text below the title stating the issue or topic of the frame, a set of numbered (or lettered) menu items of text called *selections*, and a line of standard ZOG commands called *global pads* at the bottom of the screen. (Some of these commands are: *edit*, *help*, *back*, *next*, *mark*, *return*, and *comment*.) The selections interconnect the frames. When a user selects an item by typing its number or letter at the terminal keyboard, the selected frame appears on the screen, replacing the previous frame. The structure is generally hierarchical, though cross-referencing links can be included. In addition, an item in a frame can be used to activate a process.

In 1982 ZOG was installed and used as a computer-based information management system on the nuclear-powered aircraft carrier USS CARL VINSON. This system is probably the largest and most thoroughly tested hypertext system in service in the field. ZOG has also been used for more interactive process applications such as policy analysis, authoring, communications, and code management. Historically, however, ZOG made its name more as a bulletin board/textual database/CAI tool than as an interactive system. Hence it is included in this section on browsing. A drawback of the ZOG/KMS style of viewing a single frame at a time is that users may become disoriented, since no spatial event corresponds to the process of moving from frame to frame. In the KMS system, this tendency has been offset by minimizing system response time, so that frame-to-frame transition takes about half a second. The possibility of user disorientation is greatly reduced by the fact that the user can move very quickly among frames and thus become reoriented with very little effort. Creating text and graphics is also fast in KMS.

Emacs INFO Subsystem. The help system in the widely used text editor, Emacs, is called INFO, and is much like ZOG. It has a simpler set of standard commands, and its control input is done by single letters or

short commands typed at the keyboard. It is primarily hierarchical, but a user can jump to a different place in the hierarchy by typing in the name of the destination node. It is used as an on-line help system in Emacs. INFO has the same potential for user disorientation which is shared by all of the systems which display only a single frame at a time and have no browser.

Shneiderman's Hyperties. The University of Maryland Hyperties project* has been developed in two directions—as a practical and easy-to-learn tool for browsing in instructional databases and as an experimental platform for studies on the design of hypertext interfaces. As a practical tool, it has already seen some use in the field at a Washington, D.C. museum exhibit about Austria and the Holocaust. (See Figure 6.) Designers of the exhibit emphasized making the system easy and fun for users who have never used a computer before. As an experimental platform, it has been used in five experimental studies involving over 220 subjects.¹²

In Hyperties the basic units are short articles (50-1000 words typically), which are interconnected by any number of links. The links are highlighted words or phrases in the article text. The user activates the links by touching them with a finger (on a touch-sensitive screen) or using the arrow keys to jump to them. ** Activating a link causes the article about that topic to appear in its own window on the screen. The system keeps track of the user's path through the network of articles, allowing easy return from exploratory side paths.

In addition to a title and a body of text, each article has a short (5- to 25-word) description which the program can display very quickly. This feature allows the user an intermediate position between bringing up the full article and trying to guess from the link name precisely what the article is about.

Hyperties runs on the IBM PC. Recently graphics capabilities have been added to the system. Current implementation efforts focus on support for videotisc images. Also, a browser is being developed

*The "ties" in "Hyperties" stands for "The Interactive Encyclopedia System."

**The Hyperties system uses a different convention than the mouse to select links. In the Hyperties system, some link is always selected. When the user pushes one of the arrow keys, the system responds by selecting the nearest link in the direction of the arrow. Studies showed this to be a faster and easier technique for selecting arbitrary highlighted fields on the screen.

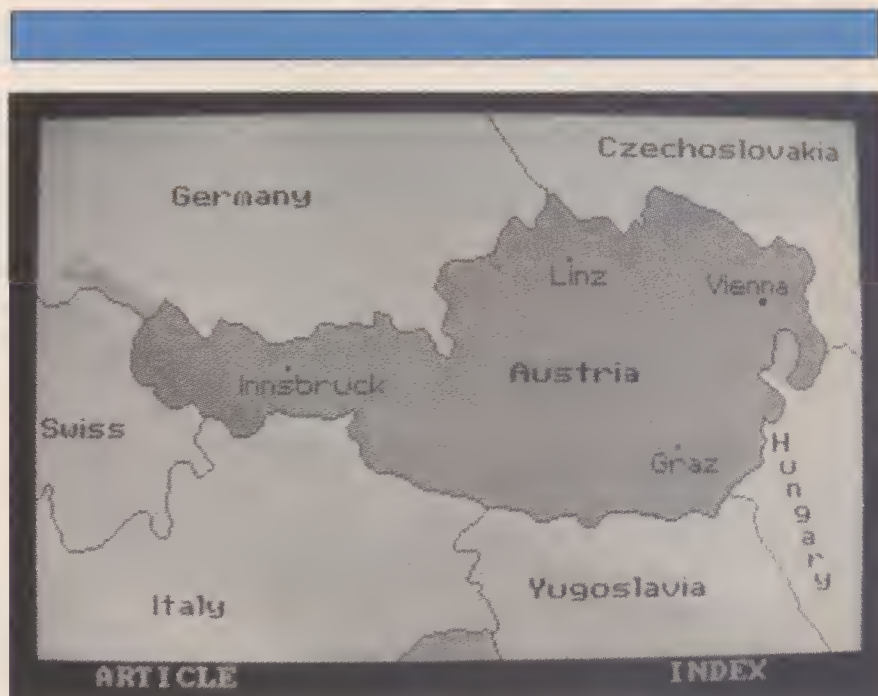
which will provide string search, bookmarks, multiple windows, and user annotation.

Symbolics Document Examiner. The most advanced of the on-line help systems, this tool displays the pages from the entire twelve-volume manual set on the Symbolics Lisp machine screen.¹³ Certain textual fields in the document (printed in bold) are mouse-sensitive. Touching one of these fields with the mouse causes the relevant section of the manual to be added to the current working set of manual pages. The system allows the reader to place *bookmarks* on any topic and to move swiftly between bookmarked topics. The protocol for link following is tailored to browsing in a reference manual or encyclopedia. Mousing a link only causes it to be placed on a list of current topics. Then, mousing an entry in this list causes that link to be followed, bringing up the referenced topic in the main viewing window.

The system also supports on-line string search of preidentified keywords, including the search for whole words, leading substrings, and embedded substrings. The system is thus well designed for the specific task of browsing through a technical manual and pursuing several aspects of a technical question or several levels of detail simultaneously. The user cannot make any changes or additions to the manual set (although it is possible to save personalized collections of bookmarks).

General hypertext technology. So far I have discussed hypertext systems that have particular practical applications. The following systems also have one or more applications, but their primary purpose is experimentation with hypertext itself as a technology. For example, while NoteCards has been used for authoring, programming, personal information management, project management, legal research, engineering design, and CAI, its developers view it primarily as a research vehicle for the study of hypertext.

Xerox PARC's NoteCards. Perhaps the best known version of full hypertext is the *NoteCards* system developed at Xerox PARC.¹⁴ The original motivation in building NoteCards was to develop an information analyst's support tool, one that would help gather information about a topic and produce analytic reports. The designers of NoteCards observed that an information analyst usually follows a general procedure that consists of a series



PLACES: AUSTRIA

PAGE 1 OF 3

Austria (see map) holds a special place in the history of the **Holocaust**.

Situated between Eastern and Western Europe, possessing a vibrant and culturally creative **Jewish community** on the eve of World War II,

Austria had also provided the young **Adolf Hitler**, himself an Austrian raised near Linz, with important lessons in the political uses of

antisemitism. Leading **Nazis** came from Austria: the names of Adolf Hitler, **Adolf Eichmann**, who organized the **deportations** of the Jews to the **death camps**, and **Ernst Kaltenbrunner**, the head of the Reich Main Office for Security, 1943-45, readily come to mind. As

Linz - city in northern Austria; childhood home of Adolf Hitler and other leading Nazis

NEXT PAGE

RETURN TO GYPSIES

INDEX

Figure 6. The Hyperties Browser enables users to traverse a database of articles and pictures by selecting from highlighted items embedded in the text of the articles. The photos show the IBM PC version of Hyperties. The upper node shows a map of Austria. The lower node shows double-spaced text with link terms highlighted. Either a touchscreen or jump-arrow keys are used for selection of brief definitions, full articles, or pictures. The Hyperties Author permits people with only word processing skills to create and maintain databases. Research versions of Hyperties run on the Enhanced Graphics Adapter to give more lines and multiple windows and on the Sun 3 workstation to show two full pages of text at a time. Current development efforts will enable readers to point at pictures and videodisc images to retrieve further information.

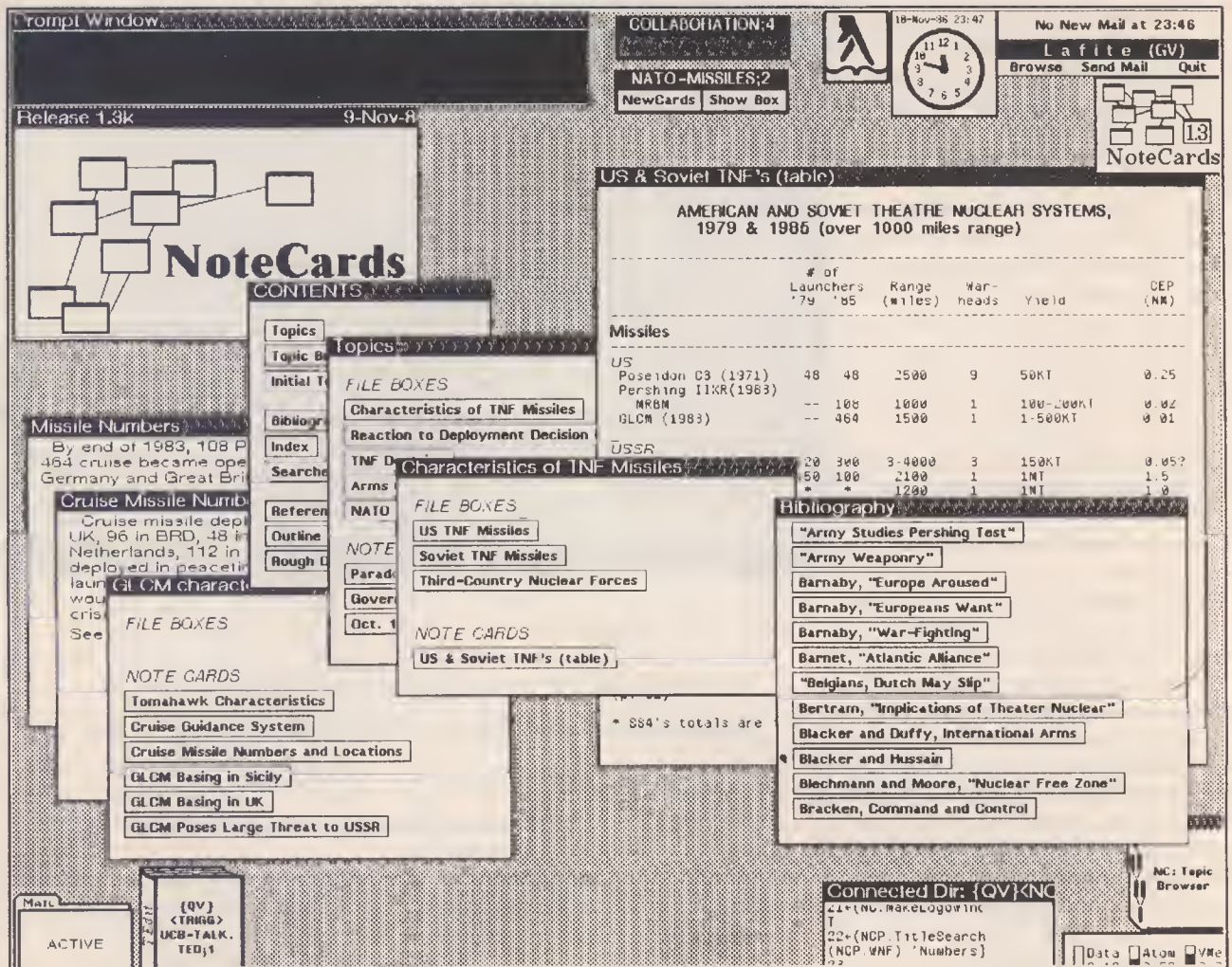


Figure 7. A typical NoteCards screen with five FileBox cards, two unformatted Text cards, and one Text card formatted as a table. Links between cards are represented by the boxed text inside the cards. The two menus at the top/middle of the screen control two different note files. The remainder of the icons on the screen belong to non-NoteCard applications running in the Xerox Lisp environment.

of steps: (1) reading sources (news reports, scholarly articles, etc.), (2) collecting clippings and filing them (in actual shoeboxes!), and (3) writing analytic reports. The designers also observed that throughout the process, the analyst forms analyses and conceptual models in his head. The research goal of the PARC team was to develop technology to aid the analyst in forming better conceptual models and analyses, and to find better expressions of these models and analyses.

A programmer's interface makes NoteCards an open architecture that allows users to build (in Lisp) new applications on top of NoteCards. Using this

interface, the user can easily customize the browser. NoteCards allows easy creation of new types of nodes. Forty or fifty such specialized node types have been created to date, including text, video, animation, graphics, and actions.* The new version also allows several users to work in the same Notefile at the same time.

Part of NoteCards' success is due to the fact that it was developed on Xerox D-series Lisp machines, which are powerful workstations that have high resolution

screens allowing windows and link and node icons to be displayed in very high resolution. (See Figure 7.) Currently between 50 and 100 users use NoteCards, many of them outside of Xerox (even though it is not a supported product). Several of these users have constructed very large databases in the system (for example, 1600 nodes with 3500 links between them).

Brown University's Intermedia. One of the oldest and largest hypertext research groups exists at Brown University, at the Institute for Research in Information and Scholarship (IRIS).¹⁵ The Intermedia

*An action node contains Lisp code which gets evaluated when a link to the node is activated.

project builds on two decades of work and three prior generations of hypertext systems.¹⁶

The first system was the Hypertext Editing System designed by Ted Nelson, Andy van Dam, and several Brown students for the IBM 2250 display in 1968. This system was used by the Houston Manned Spacecraft Center to produce Apollo documentation.

The second system was the File Retrieval and Editing System (FRESS). FRESS was a greatly enhanced multiterminal timesharing version designed by van Dam and his students. It became available in 1969 and was commercially reimplemented by Phillips in the early 1970's. FRESS was used in production by hundreds of faculty and students over more than a decade. Its users included an English poetry class that did all of its reading and writing on a communal hypertext document. Like NLS, FRESS featured both dynamic hierarchy and bidirectional reference links, and keyworded links and nodes. Unlike NLS, it imposed no limits on the sizes of nodes. On graphics terminals, multiple windows and vector graphics were supported.

The third project, the Electronic Document system, was a hypermedia system emphasizing color raster graphics and navigation aids.

As part of Brown's overall effort to bring graphics-based workstations into effective use within the classroom, the Intermedia system is being developed as a framework for a collection of tools that allow authors to create links to documents of various media such as text, timelines, diagrams and other computer-generated images, video documentaries, and music. Two courses, one on cell biology and one on English literature, have been taught using the system. Current applications include InterText, a text processor; InterDraw, a graphics editor; InterVal, a timeline editor that allows users interactively to organize information in time and date sequences; InterSpec, a viewer for sections of 3D objects; and InterPix, a scanned-image viewer. Under development are a video editor, a 2D animation editor, and more complex methods for filtering the corpus and creating and traversing trails.

Intermedia is being developed both as a tool for professors to organize and present their lesson material via computer and as an interactive medium for students to study the materials and add their own annotations and reports.

For example, in the English literature course the first time a student is searching for background information on Alexander Pope, he or she may be interested in Pope's life and the political events that prompted his satiric criticism. To pursue this line of thought the student might retrieve the biography of Pope and a timeline summarizing political events taking place in England during Pope's life. Subsequently, the student may want to compare Pope's use of satire with other later authors' satiric techniques. This time the student may look at the same information about Pope but juxtapose it with information about other satiricists instead of a time line. The instructor (and other students, if permitted) could read the student's paper, examine the reference material, and add personal annotation links such as comments, criticism, and suggestions for revision. While revising the document, the student could see all of the instructor's comments and examine the sources containing the counter-arguments.

Like most of the serious workers on hypertext, the Intermedia team is especially concerned with providing the user with ways of managing the increased complexity of the hypertext environment. For example, they contend that multiple links emanating from the same point in a document may confuse the reader. Their alternative is to have a single link icon in the material (text or graphics) which can be quickly queried via the mouse to show the specific outgoing links, their names, and their destination nodes.¹⁵ They also propose a construct called a web to implement context-dependent link display. Every link belongs to one or more webs and is only visible when one of those webs is active. To view documents with the links that belong to a particular web, a user opens a web and then opens one or more of its documents. Although other webs may also reference the document, only the links which were made in the current web are displayed. As a result, the user does not have to sift through the connections made in many different contexts.

The Intermedia project is also studying ways of providing an effective browser for a network that can include hundreds or even thousands of nodes. The Intermedia browser has two kinds of displays: a *global map*, which shows the entire hyperdocument and allows navigation within it; and a *local map*, which presents a view centered on a single document and displaying its links and nearest neighbors in the web. In addition, a display can show nodes and links at several levels of detail. For exam-

ple, it can show whole documents and the links between them, or each link and its approximate location within its documents. (See Figure 8.)

The Intermedia project has a long history, many participants, and a serious institutional commitment to long-term objectives. It conducts creative hypertext experiments and uses the classroom as a proving ground. Although this project is still in its early stages, we can expect it to contribute significantly to the development of effective cooperative work environments based on hypertext.

Tektronix Neptune. Tektronix Neptune is one hypertext system that has been particularly designed as an open, layered architecture.¹⁷ Neptune strongly separates the front end, a Smalltalk-based user interface, from the back end, a transaction-based server called the Hypertext Abstract Machine (HAM). The HAM is a generic hypertext model which provides operations for creating, modifying, and accessing nodes and links. It maintains a complete version history of each node in the hyperdocument, and provides rapid access to any version of a hyperdocument. It provides distributed access over a computer network, synchronization for multiuser access, a complex network versioning scheme, and transaction-based crash recovery.

The interface layer provides several browsers: A *graph browser* provides a pictorial view of a subgraph of nodes and links; a *document browser* supports the browsing of hierarchical structures of nodes and links; and a *node browser* accesses an individual node in a hyperdocument. Other browsers include *attribute browsers*, *version browsers*, *node differences browsers*, and *demon browsers*. (See Figure 9.)

In Neptune, each end of a link has an offset within its node, whether that node is textual or graphical.* The link attachment may refer to a particular version of a node, or it may refer to the current version. The HAM provides two mechanisms that are useful for building application layers: Nodes and links may have an unlimited number of attribute/value pairs; and special high-speed predicates are included for querying the values of these pairs in the entire hyperdocument, allow-

*Unlike in most hypertext systems, the destination end of a Neptune link is an iconic point in the text of the destination node rather than the whole node.

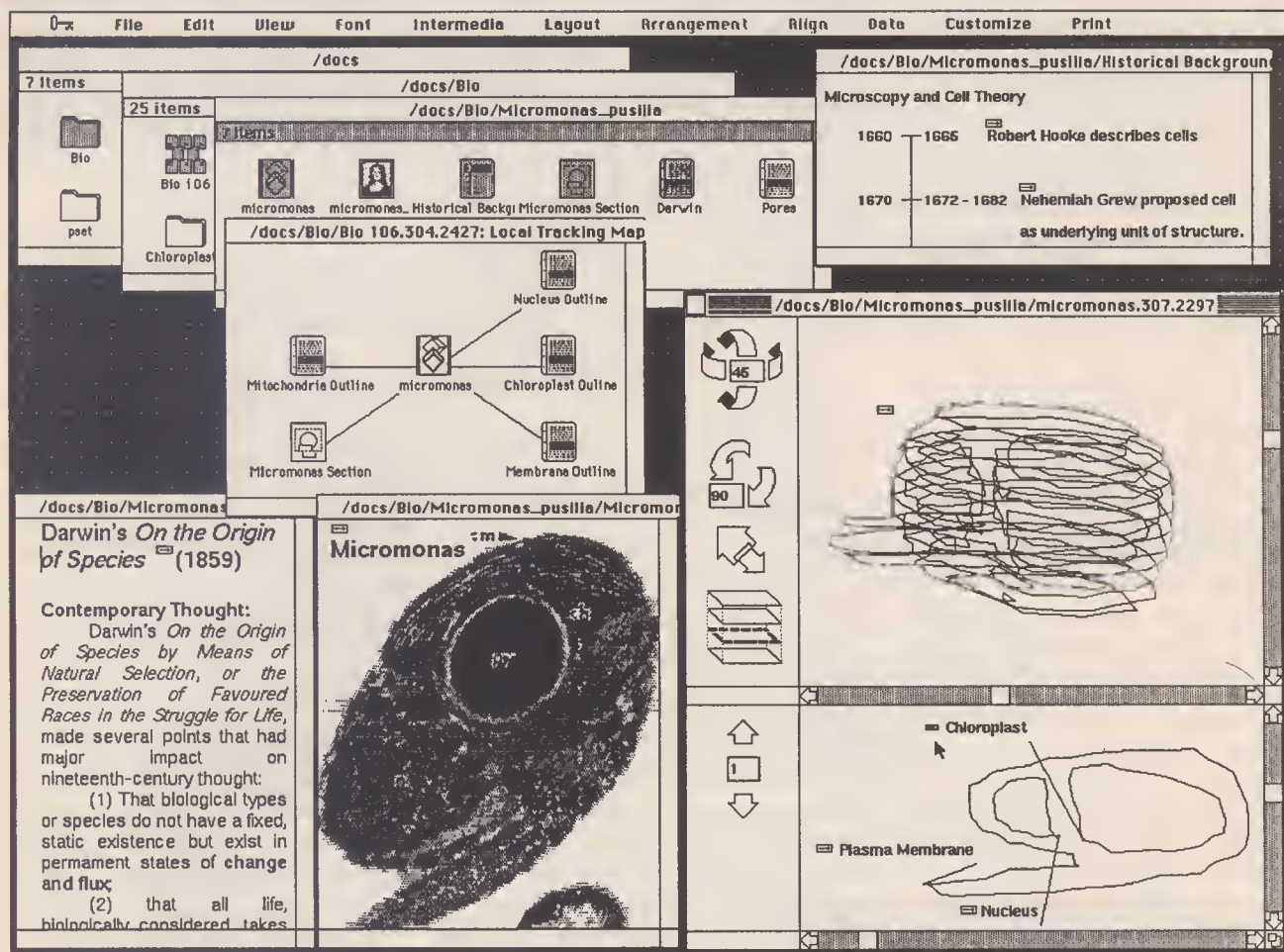


Figure 8. The Intermedia System. This figure illustrates materials from an Intermedia corpus called "Bio 106: Cell Biology in Context." Three folder windows containing hierarchically organized documents of different types are open in the upper left side of the display. An InterText document (lower left side) and an InterVal document (upper right side) are currently open, as well as an InterDraw document containing a scanned electronmicrograph (lower middle). This image has been linked to a corresponding three-dimensional image displayed in an InterSpect document (lower right). The "lower tracking map" (center) shows the links emanating from the current document. Authors or browsers can manipulate the three-dimensional image, edit text and graphics, follow links or create links at any time in this environment. (The electronmicrograph of *Micromonas* was published in the *Journal of Phycology* and is reprinted with the permission of the Editor.)

ing higher level applications to define their own accessing mechanisms on the graph. The HAM also provides a demon mechanism that invokes arbitrary code when a specific HAM event occurs.

diSessa's Boxer. Boxer¹⁸ is a highly interactive programming language specifically tailored to be easy for noncomputer specialists to learn. Boxer uses a *box* to represent a unit of information in the system. In Boxer, one box can contain other boxes, or data such as text or graphics. For

example, a program is a box that contains some boxes that provide input and output variables, and other boxes that specify behavior. The system also supports alternate views of some boxes: A box which specifies a graphics routine can also show that graphic display.

Since Boxer is a programming language, it treats cross-reference links in a special way. Rather than using mousable icons as links, Boxer uses a specialized box, called a "port," which gives a direct view into the destination. For example, a port from box

A to box B appears within A as a box which shows B. But a port is more than just a view of the destination box, because the destination box can be changed through any of the ports which lead to it, and the changes will be reflected in all of these ports.

Hierarchy is more naturally expressed in Boxer than in many of the other hypertext systems. Boxes are nested within each other two-dimensionally, and are filtered to reduce the level of clutter on the screen. This system of representation has the

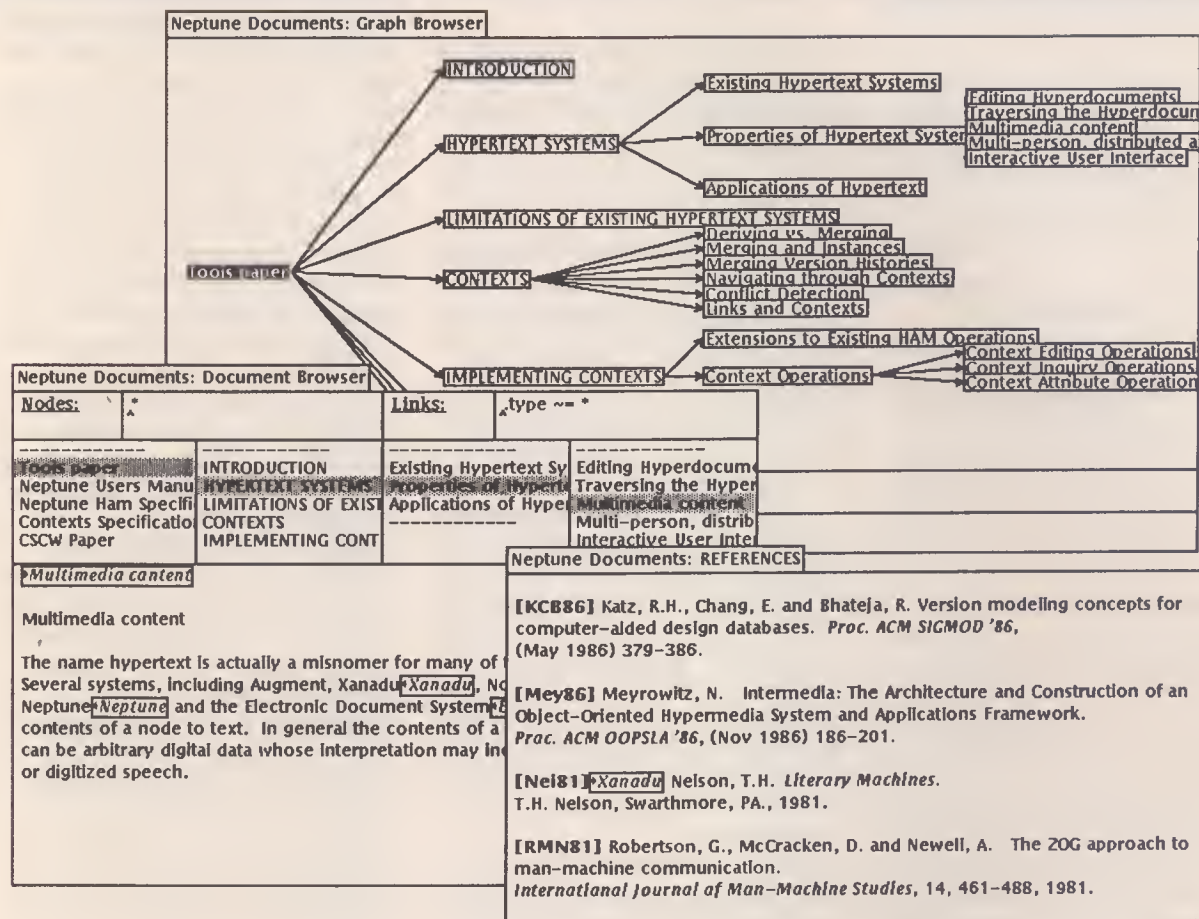


Figure 9. Neptune browsers. Three browsers from Neptune are illustrated. A pictorial view of a network of nodes and links is shown in the Graph Browser (the upper window). The lower right window and the lower pane of the Document Browser are viewers for text nodes. Icons representing link attachments are shown embedded within the text in each of the nodes.

advantage of showing a natural hierarchy of nodes: The windows of lower-level nodes are nested directly within their parents. In most hypertext systems, no attempt is made to display the parent-child relationship once the nodes are opened as windows.

Pitman's CREF. The Cross-Referenced Editing Facility (CREF) is a prototype of a specialized text and graphics editor which was developed originally as a tool for use in analyzing the transcripts from psychological experiments (known as *protocols*), but which was also used to inves-

tigate more general hypertext design issues.¹⁹ Much of the interactive feel of CREF reflects the style of use and programming of the Symbolics Lisp machine, on which it was built. Chunks of text, called *segments*, constitute the nodes in the system. Segments are arranged in linear series, and can have keywords and various kinds of links to other segments. The notion of a linear set of segments is natural to the protocol analysis problem, since the first step with such protocols is to segment them into the episodes of the experimental session.

CREF organizes segments into *collec-*

tions, which can be defined implicitly by a predicate (called an *abstract collection*) or explicitly by a list (called a *static collection*). At any time, the selected collection appears as a continuous length of text with the segment boundaries marked by named horizontal lines (such as "Segment 1," "Segment 2," etc.). This view can be edited as if it were a single document.

One way of forming an abstract collection is by selecting segments using a boolean predicate over keywords. To extend the power of this keyword facility, CREF allows the user to define a type hierarchy on the keywords. For example, if

"card 105" is defined as a type of (i.e., a child of) "card," then collections based on the keyword "card" will also contain segments which have only "card 105" as a keyword.

CREF supports four kinds of links: *references links* cross-reference among segments; *summarizes links* impose hierarchy (a *summary* is a segment which has one or more summarizes links to other segments); *supersedes links* implement versioning by copying the superseded segment and freezing it; and *precedes links* place a linear ordering on segments.

Finally, CREF allows multiple analysts to compose different *theories* about a protocol, using the same segmented data. Each theory imposes its own structure on the data, and has its own collections, diagrams, keywords, and annotations. This mode of selection is similar to the notion of contexts or webs used in other systems.

Hypertext on the Macintosh. At least two programs have been written for the Apple Macintosh that provide hypertext facilities: FileVision and Guide.

FileVision is primarily oriented to graphics nodes and to applications which can exploit visual indexing. The advertising for FileVision describes applications that encourage visual indexing. For example, in the database for a travel agency, the map of a region may contain icons for the main cities in that region. The user clicks on the icon for a city to obtain a display of a map of that city. The map of the city may have icons for the major landmarks in the city. The user clicks on one of these icons to obtain a display of data about the landmark, or perhaps even to obtain a picture of the landmark itself.

Guide is a more recent program which is based on an earlier Unix version developed in England.²⁰ It does not provide the graphics capabilities of FileVision (graphics are supported but cannot contain links), but it does support textual hypertext data very well. Guide uses three kinds of links: *replacement links*, which cause the text in the current window to be completely replaced by the text pointed to by the link; *note links*, which display the destination text in a pop-up window; and *reference links*, which bring up a new window with the destination text. Guide is now available for PCs as well.

As this article goes to press, there is news that Apple will soon have its own hypertext system, called HyperCards. HyperCards will be similar in some ways to Xerox PARC's NoteCards. It will provide

special support for *executable links*, which will give it the flavor of a programming language. HyperCards will be bundled with the system software in new Macintoshes.

MCC's PlaneText. PlaneText, developed in the MCC Software Technology Program (STP), is a very recent addition to the family of general hypertext systems. *PlaneText is based on the Unix file system and the Sun SunView window manager. Each node is a Unix file. Links appear as names in curly brackets ({}), whose display can be turned on and off. Links are implemented as pointers saved in separate files, so that the linked files themselves are not changed by creating hypertext references between them. This design allows for the smooth integration of hypertext into the rest of the Unix-based computational environment, including such tools as Mail and News. It allows for the hypertext annotation of standard source code files. In addition, the Unix file directory system serves as a "free" mechanism for creating hierarchical structures among nodes. **

PlaneText supports color graphics nodes which can be freely linked into a hyperdocument.

Summary. The systems in this section were presented in terms of four broad categories: macro literary systems, problem exploration systems, structured browsing systems, and general hypertext technology. Table I summarizes this discussion and provides a breakdown of the various features which current hypertext systems can include.

One additional area of research currently is the development of systems which aid the entire process of design, particularly the informal upstream aspects. Such systems require the features of hypertext problem exploration and structured browsing systems as well as the advanced

*It is perhaps too early to say, however, how PlaneText will rank in the world of hypertext, since it will only be publicly available from the participant companies in the MCC/Software Technology Program. For more information call Bill Stotesbery at MCC, (512) 343-0978.

**The use of an existing tree-structuring mechanism limits any hypertext system to only being able to handle a single hierarchical structure. Single hierarchical organizations may be too limited for some advanced applications.

features of the experimental hypertext technologies. Indeed, this area of investigation may become an important fifth category for hypertext systems of the future.

The history of hypertext presented here suggests that the concept and the advantages of hypertext were clear several decades ago, but that widespread interest in hypertext was delayed until the supporting technology was cheap and readily available. This suggestion may be misleading. Many of the "elders" of the field feel that something else has changed as well. They feel that today computer users easily accept the role of the computer as a tool for processing ideas, words, and symbols (in addition to numbers and mere data), and as a vehicle of interhuman communication. Those theorists who gave presentations of their hypertext systems 20 years ago, using expensive state of the art hardware, report that the computer science community showed little interest. This lack of interest seemed to stem as much from a lack of understanding of the basic concepts of hypertext as from a lack of hardware resources.

If this is so, then the recent upsurge in interest in hypertext may signal that the computer community is now ready to consider its technology as much a tool for communication and augmenting the human intellect as for analysis and information processing. Hypertext is certainly a large step in that direction.

The Essence of Hypertext

It is tempting to describe the essence of hypertext as its ability to perform high-speed, branching transactions on textual chunks. But this is a little like describing the essence of a great meal by listing its ingredients. Perhaps a better description would focus on hypertext as a computer-based medium for thinking and communication.

The thinking process does not build new ideas one at a time, starting with nothing and turning out each idea as a finished pearl. Thinking seems rather to proceed on several fronts at one, developing and rejecting ideas at different levels and on different points in parallel, each idea depending on and contributing to the others.

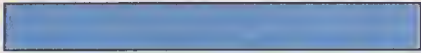
The recording and communication of such entwined lines of thought is challenging because communication is in practice

a serial process and is, in any case, limited by the bandwidth of human linguistic processing. Spoken communication of parallel themes must mark items with stresses, pauses, and intonations which the listener must remember as the speaker develops other lines of argument. Graphical forms can use lists, figures, and tables to present ideas in a less than strictly linear form. These visual props allow the reader/viewer to monitor the items which he must understand together. One of the challenges of good writing, especially good technical writing, is to present several parallel lines of a story or an argument in a way that weaves them together coherently.

Traditional flat text binds us to writing and reading paragraphs in a mostly linear succession. There are tricks for signalling branching in the flow of thought when necessary: Parenthetical comments, footnotes, intersectional references (such as "see Chapter 4"), bibliographic references, and sidebars all allow the author to say "here is a related thought, in case you are interested." There are also many rhetorical devices for indicating that ideas belong together as a set but are being presented in linear sequence. But these are rough tools at best, and often do not provide the degree of precision or the speed and convenience of access that we would like.

Hypertext allows and even encourages the writer to make such references, and allows the readers to make their own decisions about which links to follow and in what order. In this sense, hypertext eases the restrictions on the thinker and writer. It does not force a strict decision about whether any given idea is either within the flow of a paper's stream of thought or outside of it. Hypertext also allows annotations on a text to be saved separately from the reference document, yet still be tightly bound to the referent. In this sense, the "linked-ness" of hypertext provides much of its power: It is the machine processible links which extend the text beyond the single dimension of linear flow.

At the same time, some applications demonstrate that the "node-ness" of hypertext is also very powerful. Particularly when hypertext is used as a thinking, writing, or design tool, a natural correspondence can emerge between the objects in the world and the nodes in the hypertext database. By taking advantage of this object-oriented aspect, a hypertext user can build flexible networks which model his problem (or solution). In this application the links are less important



than the nodes. The links form the "glue" that holds the nodes together, but the emphasis is on the contents of the nodes.

From a computer science viewpoint, the essence of hypertext is precisely that it is a hybrid that cuts across traditional boundaries. Hypertext is a *database method*, providing a novel way of directly accessing data. This method is quite different from the traditional use of queries. At the same time, hypertext is a *representation scheme*, a kind of semantic network which mixes informal textual material with more formal and mechanized operations and processes. Finally, hypertext is an *interface modality* that features "control buttons" (link icons) which can be arbitrarily embedded within the content material by the user. These are not separate applications of hypertext: They are metaphors for a functionality that is an essential union of all three.

The power of linking. In the next two sections of this article, I will explore links and nodes in more detail as the basic building blocks of hypertext.

Link following. The most distinguishing characteristic of hypertext is its machine support for the tracing of references. But what qualifies a particular reference-tracing device as a link? How much effort is permissible on the part of a user who is attempting to trace a reference? The accepted lower limit of referencing support can be specified as follows: To qualify as hypertext, a system should require no more than a couple of keystrokes (or mouse movements) from the user to follow a single link. In other words, the interface must provide links which act like "magic buttons" to transport the user quickly and easily to a new place in the hyperdocument.

Another essential characteristic of hypertext is the speed with which the system responds to referencing requests. Only the briefest delay should occur (one or two seconds at most). Much design work goes into this feature in most systems. One reason for this concern is that the reader often does not know if he wants to pursue a link reference until he has had a cursory look at the referenced node. If making this judgement takes too long, the user may become frustrated and not bother with the hypertext links.

However, not all link traversals can be instantaneous. Perhaps as important as rapid response is providing cues to the user about the possible delay that a given query or traversal might entail. For example, some visual feature of the link icon could indicate whether the destination node is in memory, on the disk, somewhere else on the network, or archived off line.

Properties of links. Links can be used for several functions. These include the following:

- They can connect a document reference to the document itself.
- They can connect a comment or annotation to the text about which it is written.
- They can provide organizational information (for instance, establish the relationship between two pieces of text or between a table of contents entry and its section).
- They can connect two successive pieces of text, or a piece of text and all of its immediate successors.
- They can connect entries in a table or figure to longer descriptions, or to other tables or figures.

Links can have names and types. They can have a rich set of properties. Some systems allow the display of links to be turned on and off (that is, removed from the display so that the document appears as ordinary text).

The introduction of links into a text system means that an additional set of mechanisms must be added for creating new links, deleting links, * changing link names or attributes, listing links, etc.

Referential links. There are two methods for explicitly linking two points in hypertext—by reference and by organization. The reference method is a nonhierarchical method. It uses referential links that connect points or regions in the text.

Referential links are the kind of link that most clearly distinguishes hypertext. They generally have two ends, and are usually directed, although most systems support "backward" movement along the link. The origination of the link is called the "link source," and usually acts as the *reference*. The source can logically be either a single point or a region of text. At the other end, the "destination" of the link usually functions as the *referent*, and can

*Link deletion is problematical. For example, what should the policy be for nodes which are stranded when all their links have been deleted? Should they be placed in "node limbo" until the user decides what to do with them?

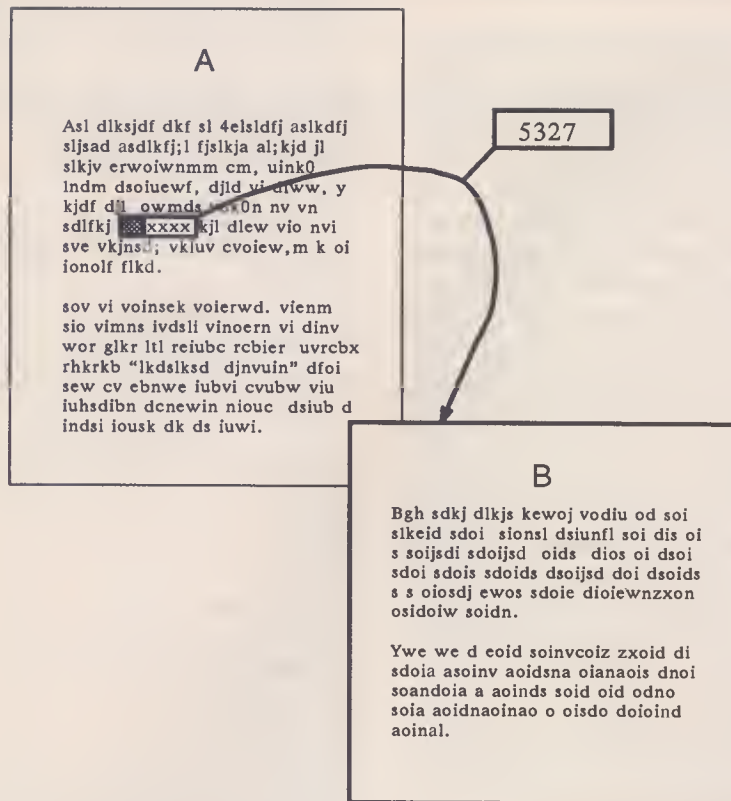


Figure 10. An example of a link with a point source and a region destination. The source of the link is a token in the text of document A which contains a textual identifier ("xxxx"). The identifier may be (1) the name of the destination node (in this case it would be "B"), (2) the name of the link, or (3) an arbitrary string which is neither the name of the link nor the destination node. The destination of this link is node B which is a region. The link has an internal name (5327) which is normally visible to the user.

also be either a point or a region. (See Figure 10.)

A *link point* is some icon indicating the presence of the link. It usually shows the link's name and perhaps also its type. Or it may show the name and/or type of the destination node. In systems such as Neptune which support links with both point source and point destination, the icon also indicates which type of link is indicated. In some systems, the display of links can be suppressed, so that the documents appear linear.

A *link region* is a set of contiguous characters which is displayed as a single unit. In Figure 10, the link destination is a link region, namely, an entire node. Fig-

ure 10 illustrates the most common form of hypertext link, in which the source is a point and the destination is a region. This example typifies many of the link applications listed above, because it shows how a chunk of text—a region—is written about or referenced by some smaller chunk of text, often a sentence. Since most readers are accustomed to single point references to sentences (i.e., footnotes), they have no problem accepting a link with a point source. There can be regions in graphics as well—either bordered regions or collections of graphic objects in a figure.

Link regions can pose difficult design problems. They are easiest to implement as whole nodes, since setting a region off

from its neighboring material within the same node raises a tough implementation issue—how to display the selected region to the user. It must be highlighted somehow, using reverse video, fonts, or color, but each of these options poses difficulties in keeping overlapping regions clearly highlighted. The Intermedia designers propose to draw a light box around regions and a darker box around region/region overlaps, thus showing a single level of overlapping¹⁵; however, this technique is not effective if there are more than two overlapping regions.

Another difficulty posed by link regions is how to show the name of the link. Unlike a link point, a link region has no obvious position for a title, unless it is placed arbitrarily at the beginning or end of the region.

Link regions can also be difficult to manipulate. Designers must devise a system for copying, moving, modifying, and deleting the region and the substrings within it. The movement of regions involves logistical dilemmas which are not easy to resolve: For example, when one moves a major portion of the text in a destination region to someplace else in the node, should the link destination move with it or stay with what remains? Also, designers must make special provisions for deleting, moving, or copying the defining end points of a region.

Organizational links. Like reference links, organizational links establish explicit links between points in hypertext. Organizational links differ from referential links in that they implement hierarchical information.

Organizational links connect a parent node with its children and thus form a strict tree subgraph within the hypertext network graph. They correspond to the *IS-A* (or *superconcept*) links of semantic net theory, and thus operate quite differently than referential links.* For example, rather than appearing as explicit highlighted tokens in each node, organizational links are often traversed by a separate mechanism at the node control level (i.e., special *goto-parent*, *goto-first-child*, and *goto-next-sibling* commands). In other cases, there are organizational nodes (such as toc nodes in Textnet and FileBoxes in NoteCards) which record the organizational structure.

*Note that organizational links are distinct from the class hierarchy links that would be used (in the object-oriented programming paradigm) to define types and subtypes of nodes in the hypertext system.

Keyword links. In addition to the explicit linking performed by referential and organizational links, there is a kind of implicit linking that occurs through the use of keywords. This type of linking is yet to be fully explored.

One of the chief advantages of text storage on a computer is the ability to search large and complex documents and sets of documents for substrings and keywords.* Naturally, this ability is also a valuable aspect of hypertext. Indeed, most users of large hyperdocuments insist on having some mechanism for scanning their content, either for selected keywords (which can apply to nodes, links, or regions) or for arbitrary embedded strings.

From a functional standpoint, link following and search are similar: Each is a way to access destination nodes that are of possible interest. Link following usually yields a single node, whereas search can yield many; hence, a keyword is a kind of implicit computed link. The value of this insight is that it may allow design of a hypertext interface which is consistent across all link-tracing activities.

To tree or not to tree. Some hypertext systems (for example, Emacs INFO) support only hierarchical structures, others (such as Xanadu and Hyperties) provide no specific support for hierarchical structures, and others (such as Textnet and NoteCards) support both kinds of structures.

One could question just how sufficient strictly hierarchical structures are, and for which applications they are sufficient and for which they are not. On the one hand, abstraction is a fundamental cognitive process, and hierarchical structures are the most natural structures for organizing levels of abstraction. On the other hand, cases obviously exist where cross-hierarchical links are required. Frank Halasz, one of the developers of NoteCards, has gathered statistics on the *hyperspace* of a single representative NoteCards user; this person had 1577 nodes (cards) in all, 502 of which were FileBoxes (hierarchical nodes). Connecting these nodes were a total of 3460 links, 2521

(73 percent) of which connected FileBoxes to each other or to individual notecards, 261 (7.5 percent) of which were nonhierarchical referential links, and the remainder of which were mail links (used by the system to tie mail messages to other nodes). This example, for what it is worth, suggests that hierarchical structure is very important in organizing a hypertext network, and that referential links are important but less common.

One advantage of a strictly tree-oriented system is that the command language for navigation is very simple: From any node, the most one can do is go to the parent, a sibling, or a child. This simplicity also diminishes the disorientation problem, since a simpler cognitive model of the information space will suffice.

Of course, the great disadvantage of any hierarchy is that its structure is a function of the few specific criteria that were used in creating it. For example, if one wishes to investigate what sea-based life forms have in common with land-based life forms, one may find that the traditional classification of life forms into the plant and animal kingdoms breaks up the information in the wrong way. The creator of a hierarchical organization must anticipate the most important criteria for later access to the information. One solution to this dilemma is to allow the information elements to be structured into multiple hierarchies, thus allowing the world to be "sliced up" into several orthogonal decompositions. Any hypertext system which has hierarchy nodes, such as Textnet (toc nodes) and NoteCards (FileBox nodes), can perform this operation quite easily. These are the only systems which explicitly claim to support multiple hierarchies. Indeed, one early user of NoteCards used the system in doing the research and writing for a major project paper; he imposed one organization on the data and his writings while doing the research, and then quite a different (yet coexistent) organization on the same material to produce his paper. As a generalization, it seems that engineering-oriented hypertext users prefer hierarchical organizations, whereas arts- or humanities-oriented users prefer cross-referencing organizations.

Extensions to basic links. Certain features of the link enable it to be extended in

several ways. Links can connect more than two nodes to form *cluster links*. Such cluster links can be useful for referring to several annotations with a single link, and for providing specialized organizational structures among nodes. Indeed, the toc nodes of Textnet and the FileBoxes of NoteCards are both forms of cluster links.

One useful way to extend the basic link is to place attribute/value pairs on links and to query the network for them. The Neptune system, for example, has an architecture that is optimized for this function. Coupled with specialized routines in the database interpreter (the HAM), these attribute lists allow users to customize links in several ways, including devising their own type system for links and performing high-speed queries on the types.

It is also possible to perform procedural attachments on a link so that traversing the link also performs some user-specified side effect, such as customizing the appearance of the destination node. This ability is provided in Neptune and Boxer.

Hypertext nodes. Although the essence of hypertext is its machine-supported linking, the nodes contribute significantly to defining the operations that a hypertext system can perform. Most users of hypertext favor using nodes which express a single concept or idea, and are thus much smaller than traditional files. When nodes are used in this fashion, hypertext introduces an intermediate level of machine support between characters and files, a level which has the vaguely semantic aspect of being oriented to the expression of ideas. But this sizing is completely at the discretion of the hypertext writer, and the process of determining how to modularize a document into nodes is an art, because its impact on the reader is not well understood.²¹

The modularization of ideas. Hypertext invites the writer to modularize ideas into units in a way that allows (1) an individual idea to be referenced elsewhere, and (2) alternative successors of a unit to be offered to the reader (for instance, more detail, an example, bibliographic references, or the logical successor). But the writer must also reckon with the fact that a hypertext node, unlike a textual paragraph, tends to be a strict unit which does not blend seamlessly with its neighbors. Some hypertext systems (Notecards, CREF, Boxer, FRES, NLS) allow nodes to be viewed together as if they were one big node, and this option is essential for some

*There is some controversy over the relative merits of keyword retrieval as opposed to full text search. On the one hand, keyword retrieval is only as good as the skill and thoroughness of the person selecting the keywords. On the other hand, full text search does not find all the relevant documents, nor does it always find only the relevant documents. Its shortcomings are due in part to the commonness of synonyms in English. In addition, full text search can be computationally prohibitive in large networks.

applications (for example, writing and reading prose). But the boundaries around nodes are always discrete and require sometimes difficult judgements about how to cleave the subject matter into suitable chunks.

The process of identifying a semantically based unit, such as an idea or concept, with a syntactic unit, such as a paragraph or hypertext node, is not unique to hypertext. Manuals of style notwithstanding, traditional text has rather loose conventions for modularizing text into paragraphs. This looseness is acceptable because paragraph boundaries have a relatively minor effect on the flow of the reading. Paragraph boundaries are sometimes provided just to break up the text and give the eye a reference point. Thus, decisions about the distribution of sentences among paragraphs is not always critical.

Hypertext, on the other hand, can enforce a rather stern information hiding. In some systems, the only clue a user has as to the contents of a destination node is the name of the link (or the name of the node, if that is provided instead). The writer is no longer making all the decisions about the flow of the text. The reader can and must constantly decide which links to pursue. In this sense, hypertext imposes on both the writer and the reader the need for more process awareness, since either one has the option of *branching* in the flow of the text. Thus hypertext is best suited for applications which require these kinds of judgements anyway, and hypertext merely offers a way to act directly on these judgements and see the results quickly and graphically.

Ideas as objects. While difficult to document, there is something very compelling about reifying the expression of ideas into discrete objects to be linked, moved, and changed as independent entities. Alan Kay and Adele Goldberg²² observed of Smalltalk that it is able to give objects a perceptual dimension by allocating to them a rectangular piece of screen real estate. This feature offers enhanced retrieval and recognition over computer-processed flat documents, because to a much greater degree abstract objects are directly associated with perceptual objects—the windows and icons on the screen.

Paragraphs, sections, and chapters in a book, viewed through a standard text editor or word processor, don't stand out as first-class entities. This is particularly apparent when one can view one's docu-

ment hierarchically (i.e., as an outline) at the same time that one adds new sections and embellishes existing ones. People don't think in terms of "screenfuls"; they think in terms of ideas, facts, and evidence. Hypertext, via the notion of nodes as individual expressions of ideas, provides a vehicle which respects this way of thinking and working.

Typed nodes. Some hypertext systems sort nodes into different types. These *typed nodes* can be extremely useful, particularly if one is considering giving them some internal structure, since the types can be used to differentiate the various structural forms.

For example, in our research in the MCC Software Technology Program, we have been implementing a hypertext interface for a design environment called the Design Journal. The Design Journal is intended to provide an active scratchpad in which the designer can deliberate about design decisions and rationale, both individually and in on-line design meetings, and in which he can integrate the design itself with this less formal kind of information. For this purpose we have provided a set of four typed nodes for the designer to use—*notes*, *goals/constraints*, *artifacts*, and *decisions*. Notes are used for everything from reminders, such as "Ask Bill for advice on Module X," to specific problems and ideas relating to the design. Goals/constraints are for the initial requirements as well as discovered constraints within the design. Artifacts are for the elements of the output: The Design. And decisions are for capturing the branch points in the design process, the alternatives considered by the designer, and some of the rationale for any commitment (however tentative) that has been made. The designer captures assumptions in the form of decisions with only one alternative. Our prototype of the Design Journal uses color to distinguish between note types in the browser, and we have found this to be a very effective interface.

Hypertext systems that use typed nodes generally provide a specialized color, size, or iconic form for each node type. The distinguishing features help the user differentiate at a glance the broad classes of typed nodes that he is working with. Systems such as NoteCards, Intermedia, and IBIS make extensive use of typed nodes.

Semistructured Nodes. So far I have spoken of the hypertext node as a structureless "blank slate" into which one might put a word or a whole document. For some applications, there is growing interest in *semistructured nodes*—typed nodes which contain labelled fields and spaces for field values. The purpose of providing a template for node contents is to assist the user in being complete and to assist the computer in processing the nodes. The less that the content of a node is undifferentiated natural language (for example, English) text, the more likely that the computer can do some kinds of limited processing and inference on the textual subchunks. This notion is closely related to Malone's notion of semistructured information systems.²³

To continue with the example of the Design Journal, we have developed a model for the internal structure of decisions. The model is named ISAAC. It assumes that there are four major components to a design decision:

- (1) an *issue*, including a short name for the issue and a short paragraph describing it in general terms;
- (2) a set of *alternatives*, each of which resolves the issue in a different way, each having a name and short description, and each potentially linked to the design documents or elements that implement the alternative;
- (3) an *analysis* of the competing alternatives, including the specific criteria being used to evaluate them, the trade-off analysis among these alternatives, and links to any data that the analysis draws upon; and
- (4) a *commitment* to one of the alternatives (however tentatively) or to a vector of preferences over the alternatives, and a subjective rating about the correctness or confidence of this commitment.

Without getting into the details of the underlying theory, I merely wish to stress here that the internal structure of ISAAC suggests that the author of an ISAAC decision is engaged in a much more structured activity than just "writing down the decision," and the reader is likewise guided by the regularity of the ISAAC structure.

Of course, it may not be clear why we do not treat each of the elements listed above as its own typed hypertext node. The reason is that the parts of an ISAAC frame are much more tightly bound together than ISAAC frames are bound to each other. For example, we could not have an

analysis part without an alternatives part; yet if we treat them as separate hypertext nodes, we have failed to build this constraint into the structure. The general issue here is that some information elements must always occur together, while others may occur together or not, depending on how related they are in a given context and how important it is to present them as a cluster distinct from "surrounding" information elements. This problem is recursive: An element that is atomic at one level may turn out on closer inspection to contain many components, some of which are clustered together.

In hypertext this tension presents itself as the twin notions of semistructured nodes and composite nodes.

Composite nodes. Another mechanism for aggregating related information in hypertext is the *composite node*. Several related hypertext nodes are "glued" together and the collection is treated as a single node, with its own name, types, versions, etc. Composite nodes are most useful for situations in which the separate items in a bulleted list or the entries in a table are distinct nodes but also cohere into a higher level structure (such as the list or table). This practice can, however, undermine the fundamental association of one interface object (window) per database object (node), and thus must be managed well to avoid complicating the hypertext idiom unduly.

A composite node facility allows a group of nodes to be treated as a single node. The composite node can be moved and resized, and closes up to a suitable icon reflecting its contents. The subnodes are separable and rearrangeable through a subedit mode. The most flexible means of displaying a composite node is to use a constraint language (such as that developed by Symbolics for Constraint Frames) which describes the subnodes as *panes* in the composite node window and specifies the interpane relationships as dynamic constraints on size and configuration.

Composite nodes can be an effective means of managing the problem of having a large number of named objects in one's environment. Pitman described the problem this way:

In this sort of system, there is a never-ending tension between trying to name everything (in which case, the number of named things can grow quickly and the set can become quickly unmanageable) or to name as little as possible (in which case, things that took a lot of trouble to construct can be hard to retrieve if one accidentally drops the pointers to them).¹⁹



One problem with composite nodes is that as the member nodes grow and change the aggregation can become misleading or incorrect. A user who encounters this problem is in the same predicament as a writer who has rewritten a section of a paper so thoroughly that the section title is no longer accurate. This "semantic drift" can be difficult to catch.

Analogy to semantic networks. The idea of building a directed graph of informal textual elements is similar to the AI concept of *semantic networks*. A semantic network is a knowledge representation scheme consisting of a directed graph in which concepts are represented as nodes, and the relationships between concepts are represented as the links between them. What distinguishes a semantic network as an AI representation scheme is that concepts in the representation are indexed by their semantic content rather than by some arbitrary (for example, alphabetical) ordering. One benefit of semantic networks is that they are natural to use, since related concepts tend to cluster together in the network. Similarly, an incompletely or inconsistently defined concept is easy to spot since a meaningful context is provided by those neighboring concepts to which it is already linked.

The analogy to hypertext is straightforward: Hypertext nodes can be thought of as representing single concepts or ideas, internode links as representing the semantic interdependencies among these ideas, and the process of building a hypertext network as a kind of informal knowledge engineering. The difference is that AI knowledge engineers are usually striving to build representations which can be mechanically interpreted, whereas the goal of the hypertext writer is often to capture an interwoven collection of ideas without regard to their machine interpretability. The work on semantic networks also suggests some natural extensions to hypertext, such as typed nodes, semistructured nodes (frames), and inheritance hierarchies of node and link types.

The advantages and uses of hypertext

Intertextual references are not new. The

importance of hypertext is simply that references are machine-supported. Like hypertext, traditional literature is richly interlinked and is hierarchically organized. In traditional literature, the medium of print for the most part restricts the flow of reading to follow the flow of linearly arranged passages. However, the process of following side links is fundamental even in the medium of print. In fact, library and information science consist principally of the investigation of side links. Anyone who has done research knows that a considerable portion of that effort lies in obtaining referenced works, looking up cross-references, looking up terms in a dictionary or glossary, checking tables and figures, and making notes on notecards. Even in simple reading one is constantly negotiating references to other chapters or sections (via the table of contents or references embedded in the text), index entries, footnotes, bibliographic references, sidebars, figures, and tables. Often a text invites the reader to skip a section if he is not interested in greater technical detail.

But there are problems with the traditional methods.

- Most references can't be traced backwards: A reader can not easily find where a specific book or article is referenced in a document, nor can the author of a paper find out who has referenced the paper.

- As the reader winds his way down various reference trails, he must keep track of which documents he has visited and which he is done with.

- The reader must squeeze annotations into the margins or place them in a separate document.

- Finally, following a referential trail among paper documents requires substantial physical effort and delays, even if the reader is working at a well-stocked library. If the documents are on line, the job is easier and faster, but no less tedious.

New possibilities for authoring and design.

Hypertext may offer new ways for authors and designers to work. Authoring is usually viewed as a word- and sentence-level activity. Clearly the word processor*

*Actually, the term "word processor" is quite misleading. Most such tools accept input only at the character level, and manipulate characters, words, sentences, and paragraphs with equal facility. So these tools manipulate units of text, not words. But do they "process" these units? "Processing" implies that the computer performs some additional work, such as changing the verb form if the subject was changed from singular to plural, or performing real-time spelling and grammar correction. Since this is not the case, we really should return to the original term for these tools: "text editors".

is a good tool for authoring at this level. However, authoring obviously has much to do with structuring of ideas, order of presentation, and conceptual exploration. Few authors simply sit down and pour out a finished text, and not all editing is just "wordsmithing" and polishing. In a broad sense, authoring is the *design of a document*. The unit of this level of authoring is the idea or concept, and this level of work can be effectively supported by hypertext, since the idea can be expressed in a node. As the writer thinks of new ideas, he can develop them in their own nodes, and then link them to existing ideas, or leave them isolated if it is too early to make such associations. The specialized refinements of a hypertext environment assist the movement from an unstructured network to the final polished document.

New possibilities for reading and retrieval. Hypertext may also offer new possibilities for accessing large or complex information sources. A linear (nonhypertext) document can only be easily read in the order in which the text flows in the book. The essential advantage of nonlinear text is the ability to organize text in different ways depending on differing viewpoints. Shasha provides the following description of this advantage:

Suppose you are a tourist interested in visiting museums in a foreign city. You may be interested in visual arts. You may want to see museums in your local area. You may only be interested in inexpensive museums. You certainly want to make sure the museums you consider are open when you want to visit them. Now your guidebook may be arranged by subject, by name of museum, by location, and so on. The trouble is: if you are interested in any arrangement other than the one it uses, you may have to do a lot of searching. You are not likely to find all the visual arts museums in one section of a guidebook that has been organized by district. You may carry several guidebooks, each organized by a criterion you may be interested in. The number of such guidebooks is a measure of the need for a nonlinear text system.²¹

Another advantage is that it is quite natural in a hypertext environment to suspend reading temporarily along one line of investigation while one looks into some detail, example, or related topic. Bush described an appealing scenario in his 1945 article:

The owner of the memex, let us say, is interested in the origin and properties of the bow and arrow. Specifically he is studying why the short Turkish bow was apparently superior to the English long bow in the skirmishes of the Crusades. He has dozens of possibly pertinent books and articles in his

memex. First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes, building a trail of many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a [permanent] trail of his interest through the maze of materials available to him.²

As we have seen, Bush's notion of the "trail" was a feature of Trigg's Textnet,⁶ allowing the hypertext author to establish a mostly linear path through the document(s). The main (default) trail is well marked, and the casual reader can read the text in that order without troubling with the side trails.

Summary. We can summarize the operational advantages of hypertext as:

- *ease of tracing references*: machine support for link tracing means that all references are equally easy to follow forward to their referent, or backward to their reference;
- *ease of creating new references*: users can grow their own networks, or simply annotate someone else's document with a comment (without changing the referenced document);
- *information structuring*: both hierarchical and nonhierarchical organizations can be imposed on unstructured information; even multiple hierarchies can organize the same material;
- *global views*: browsers provide table-of-contents style views, supporting easier restructuring of large or complex documents; global and local (node or page) views can be mixed effectively;
- *customized documents*: text segments can be threaded together in many ways, allowing the same document to serve multiple functions;
- *modularity of information*: since the same text segment can be referenced from several places, ideas can be expressed with less overlap and duplication;
- *consistency of information*: references are embedded in their text, and

if the text is moved, even to another document, the link information still provides direct access to the reference;

- *task stacking*: the user is supported in having several paths of inquiry active and displayed on the screen at the same time, such that any given path can be unwound to the original task;
- *collaboration*: several authors can collaborate, with the document and comments about the document being tightly interwoven (the exploration of this feature has just begun).

The disadvantages of hypertext

There are two classes of problems with hypertext: problems with the current implementations and problems that seem to be endemic to hypertext. The problems in the first class include delays in the display of referenced material, restrictions on names and other properties of links, lack of browsers or deficiencies in browsers, etc. The following section outlines two problems that are more challenging than these implementation shortcomings, and that may in fact ultimately limit the usefulness of hypertext: *disorientation* and *cognitive overhead*.

Getting "lost in space." Along with the power to organize information much more complexly comes the problem of having to know (1) where you are in the network and (2) how to get to some other place that you know (or think) exists in the network. I call this the "disorientation problem." Of course, one also has a disorientation problem in traditional linear text documents, but in a linear text, the reader has only two options: He can search for the desired text earlier in the text or later in the text. Hypertext offers more degrees of freedom, more dimensions in which one can move, and hence a greater potential for the user to become lost or disoriented. In a network of 1000 nodes, information can easily become hard to find or even forgotten altogether. (See Figure 11.)

There are two major technological solutions for coping with disorientation—graphical browsers and query/search mechanisms. Browsers rely on the extremely highly developed visuospatial processing of the human visual system. By placing nodes and links in a two- or three-dimensional space, providing them with

done on effective and standardized methods for ellision.

The cognitive task scheduling problem.

The other fundamental problem with using hypertext is that it is difficult to become accustomed to the additional mental overhead required to create, name, and keep track of links. I call this "cognitive overhead." Suppose you are writing about *X*, and a related thought about *Y* comes to mind and seems important enough to capture. Ideally, hypertext allows you to simply "press a button" (using some mouse or keyboard action) and a new, empty hypertext window pops onto the screen. You record *Y* in this new window, then you press another button, the *Y* window disappears, and you are in the *X* window right where you were when *Y* occurred to you.

Unfortunately, the situation is a bit more complex than this scenario implies. If *Y* has just occurred to you, it may still be hazy and tentative; the smallest interruption could cause you to lose it. Coming up with a good word or short phrase to summarize *Y* may not be easy. You have to consider not just what is descriptive but also what will be suggestive for the reader when he encounters the link to *Y* within *X*. In addition, you must determine whether you should name the link to *Y* to suggest the contents of *Y* or to show *Y*'s relationship to *X*. Some systems (for example, NoteCards) provide that links can have both a *type* (such as "idea") and a *label* (such as "subsume *A* in *B*"). Coming up with good names for both can impose even more load on an author struggling with an uncertain point. (One way to reduce this problem is for the authoring system to support immediate recording of the substance of the idea, deferring the creation and labeling of the link and/or the node until after the thought has been captured.)

Beyond that, you must also consider if you have provided sufficient links to *Y* before returning to work on *X*. Perhaps there are better ways to link *Y* to the network of thoughts than at the point in *X* where *Y* came to mind.

The problem of cognitive overhead also occurs in the process of reading hypertext, which tends to present the reader with a large number of choices about which links to follow and which to leave alone. These choices engender a certain overhead of metalevel decision making, an overhead that is absent when the author has already made many of these choices for you. At the moment that you encounter a link,

how do you decide if following the side path is worth the distraction? Does the label appearing in the link tell you enough to decide? This dilemma could be called "informational myopia." The problem is that, even if the system response time is instantaneous (which it rarely is), you experience a definite distraction, a "cognitive loading," when you pause to consider whether to pursue the side path. This problem can be eased by (1) having the cross-referenced node appear very rapidly (which is the approach of KMS), (2) providing an instantaneous one- to three-line explanation of the side reference in a pop-up window (which is the approach of Intermedia), and (3) having a graphical browser which shows the local subnetwork into which the link leads.

These problems are not new with hypertext, nor are they mere byproducts of computer-supported work. People who think for a living—writers, scientists, artists, designers, etc.—must contend with the fact that the brain can create ideas faster than the hand can write them or the mouth can speak them. There is always a balance between refining the current idea, returning to a previous idea to refine it, and attending to any of the vague "proto-ideas" which are hovering at the edge of consciousness. Hypertext simply offers a sufficiently sophisticated "pencil" to begin to engage the richness, variety, and interrelatedness of creative thought. This aspect of hypertext has advantages when this richness is needed and drawbacks when it is not.

To summarize, then, the problems with hypertext are

- *disorientation*: the tendency to lose one's sense of location and direction in a nonlinear document; and
- *cognitive overhead*: the additional effort and concentration necessary to maintain several tasks or trails at one time.

These problems may be at least partially resolvable through improvements in performance and interface design of hypertext systems, and through research on information filtering techniques.

In this article, I have reviewed existing hypertext systems, the opportunities and problems of hypertext, and some of the top-level design issues of building hypertext systems. It has been my intention to give the reader a clear sense of what hypertext is, what its strengths and weaknesses are, and what it can be used

for. But I also intended something more: that the reader come away from this article excited, eager to try using hypertext for himself, and aware that he is at the beginning of something big, something like the invention of the wheel, but something that still has enough rough edges that no one is really sure that it will fulfill its promise.

To that end, I mention one more book that might be considered to belong to the literature on hypertext. *Neuromancer*²⁴ is a novel about a time in the distant future when the ultimate computer interface has been perfected: One simply plugs one's brain into the machine and experiences the computer data directly as perceptual entities. Other computers look like boxes floating in three-dimensional space, and passwords appear as various kinds of doors and locks. The user is completely immersed in a virtual world, the "operating system," and can move around and take different forms simply by willing it.

This is the ultimate hypertext system. The basic idea of hypertext, after all, is that ideas correspond to perceptual objects, and one manipulates ideas and their relationships by directly manipulating windows and icons. Current technology limits the representation of these objects to static boxes on a CRT screen, but one can easily predict that advances in animation, color, 3D displays, sound, etc.—in short, Nelson's *hypermedia*—will keep making the display more active and realistic, the data represented richer and more detailed, and the input more natural and direct. Thus, hypertext, far from being an end in itself, is just a crude first step toward the time when the computer is a direct and powerful extension of the human mind, just as Vannevar Bush envisioned when he introduced his Memex four decades ago. □

Acknowledgements

I wish to thank Les Belady, Bill Curtis, Susan Gerhart, Raymonde Guindon, Eric Gullichsen, Frank Halasz, Peter Marks, and Andy van Dam for their thoughtful reading of previous drafts.

References

1. T.H. Nelson, "Getting It Out of Our System," *Information Retrieval: A Critical Review*, G. Schechter, ed., Thompson Books, Wash., D.C., 1967.
2. V. Bush, "As We May Think," *Atlantic Monthly*, July 1945, pp. 101-108.
3. D.C. Engelbart, "A Conceptual Framework for the Augmentation of Man's

Intellect," in *Vistas in Information Handling*, Vol. 1, Spartan Books, London, 1963.

4. D.C. Engelbart and W.K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conf. Proc.*, Vol. 33, Part 1, The Thompson Book Company, Washington, D.C., 1968.

5. T.H. Nelson, "Replacing the Printed Word: A Complete Literary System," *IFIP Proc.*, October 1980, pp. 1013-1023.

6. R.H. Trigg, *A Network-based Approach to Text Handling for the Online Scientific Community*, PhD. Thesis, University of Maryland, 1983.

7. H. Rittel and M. Webber, "Dilemmas in a General Theory of Planning," *Policy Sciences*, Vol. 4, 1973.

8. D.G. Lowe, "Cooperative Structuring of Information: The Representation of Reasoning and Debate," in *Int'l. J. of Man-Machine Studies*, Vol. 23, 1985, pp. 97-111.

9. J.B. Smith et al, "WE: A Writing Environment for Professionals," Technical Report 86-025, Department of Computer Science, University of North Carolina at Chapel Hill, August 1986.

10. W. Hershey, "Idea Processors," *BYTE*, June 1985, p. 337.

11. D. McCracken and R.M. Akscyn, "Experience with the ZOG Human-computer Interface System," *Int'l. J. of Man-Machine Studies*, Vol. 21, 1984, pp. 293-310.

12. B. Shneiderman and J. Morariu, "The Interactive Encyclopedia System (TIES)," Department of Computer Science, University of Maryland, College Park, MD 20742, June 1986.

13. J.H. Walker, "The Document Examiner," *SIGGRAPH Video Review*, Edited Compilation from CHI'85: *Human Factors in Computing System*, 1985.

14. F.G. Halasz, T.P. Moran, and T.H. Trigg, "NoteCards in a Nutshell," *Proc. of the ACM Conf. on Human Factors in Computing Systems*, Toronto, Canada, April 1987.

15. N.L. Garrett, K.E. Smith, and N. Meyrowitz, "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System," in *Proc. Conf. on Computer-Supported Cooperative Work*, MCC Software Technology Program, Austin, Texas, 1986.

16. N. Yankelovich, N. Meyrowitz, and A. van Dam, "Reading and Writing the Electronic Book," *Computer*, October 1985.

17. N. Delisle and M. Schwartz, "Neptune: A Hypertext System for CAD Applications," *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, Washington, D.C., May 28-30, 1986, pp. 132-143. (Also available as SIGMOD Record Vol. 15, No. 2, June 1986).

18. A. diSessa, "A Principled Design for an Integrated Computational Environment," *Human-Computer Interaction*, Vol. 1, Lawrence Erlbaum, 1985, pp. 1-47.

19. K.M. Pitman, "CREF: An Editing Facility for Managing Structured Text," A.I. Memo No. 829, M.I.T. A.I. Laboratory, Cambridge, Mass., February 1985.

20. P.J. Brown, "Interactive Documentation," in *Software: Practice and Experience*, March 1986, pp. 291-299.

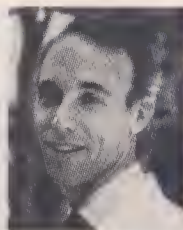
21. D. Shasha, "When Does Non-Linear Text Help? *Expert Database Systems*, *Proc. of the First Int'l Conf.*, April 1986, pp. 109-121.

22. A. Kay and A. Goldberg, "Personal Dynamic Media," *Computer*, March 1977, pp. 31-41.

23. T.W. Malone et al, "Intelligent Information-Sharing Systems," *Communications of the ACM*, May 1987, pp. 390-402.

24. W. Gibson, *Neuromancer*, Ace Science Fiction, 1984.

A more detailed version of this article, including an extended bibliography, is available from the author. To obtain a copy, circle number 181 on the Reader Service Card at the back of the magazine.



E. Jeffrey Conklin is a member of the research staff and GE's liaison to the Software Technology Program in the Microelectronics and Computer Technology Corporation (MCC). His research centers on constructing information systems for the capture and use of design rationale.

Conklin Received his BA from Antioch College and his MS and PhD from the University of Massachusetts at Amherst.

Readers may write to Conklin at MCC Software Technology Program, P.O. Box 200195, Austin, TX 78720; (512) 343-0978.

CALL FOR PAPERS

RIA0 88

User-Oriented Content-Based Text and Image Handling

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge, MA ■ March 21-24, 1988

General Theme: Information retrieval systems for full-text and mixed media databases handling:

- techniques designed to reduce imprecision in full-text database searching
- data entry and control
- "friendly" end-user interfaces
- new media

Specific Themes:

- A) Linguistic processing and interrogation of full-text databases.
- B) Automatic thesaurus construction.
- C) Expert system techniques for retrieving information in full-text and multimedia databases.
- D) Friendly user interfaces to classical information retrieval systems.
- E) Specialized machines and system architecture designed for treating full-text data, including managing and accessing widely distributed databases.
- F) Automatic database construction using scanning techniques, optical character readers, output document preparation, etc . . .
- G) New applications and perspectives suggested by emerging new technologies.

DEADLINES:

SEPTEMBER 30, 1987: *Intention of submitting a paper.*

OCTOBER 30, 1987: *Four copies should be received by the Conference Secretariat.*

DECEMBER 16, 1987: *Acceptance notification.*

CONFERENCE ORGANIZED BY: Centre National de la Recherche Scientifique (CNRS), Centre National de Recherche des Télécommunications (CNET), Institut National de Recherche en Informatique et Automatique (INRIA), Ecole Nationale Supérieure des Mines de Paris, Centre de Hautes Etudes Internationales d'Informatique Documentaires (CID). U.S. participating organizations: American Federation of Information Processing Societies (AFIPS), American Society for Information Sciences (ASIS), Information Industry Association (IIA)

Submit papers to, or for more information:

RIA0 88 Conference Service Office, MIT, Bldg. 7, Rm. 111, Cambridge, MA 02139 USA

Software Architects

Join a unique start-up with influential friends.

MAS is a well-funded California start-up financed by a highly successful Silicon Valley corporation and two notable joint-development partners — IBM and Ford Motor Company.

We are currently seeking the following experienced professionals to join our rapidly growing advanced product development team, and build a family of CIM system software products for discrete manufacturing.

Application Generator Architect

Key role in the development of a graphical 4GL for CIM applications in an OS/2 environment. A background in application generation and interpreted languages is required.

User Interface Specialist

Lead position in the development of highly ergonomic graphical interfaces for the application generator and runtime environment. You will be working in an OS/2 VGA environment. Strong background in user interfaces is required.

Object-Oriented System Specialist

Key role in designing, coding and testing of modules for a new objected-oriented CIM production control system. Knowledge of

OS internals, compilers, linkers, dynamic loaders, relational DBMS and distributed real-time systems a plus.

Application Developer

Key position in designing of object-oriented application modules for a discrete manufacturing production control system. Experience with production control applications, objected oriented concepts, on-line transaction processing and relational DBMS a plus.

Application Specialist

Internal consultant providing key input into the design of CIM application generation tools. You will have a key influence on the product direction. Must have significant background in implementation of discrete plant floor applications.

All positions require a BSCS or equivalent and at least 5 years' relevant experience. Knowledge of C is desirable. These opportunities are located in the San Francisco Bay Area.

In addition to start-up equity participation, we provide the competitive benefits offered by larger corporations.

To apply, please send your resume to Professional Staffing, Measurex Automation Systems, Dept. IEEE, 10411 Bubb Road, Cupertino, CA 95014. We are an equal opportunity employer.

mas MEASUREX
AUTOMATION
SYSTEMS

A UNIQUE START-UP

Improving Software Productivity

Barry W. Boehm, TRW

Computer hardware productivity continues to increase by leaps and bounds, while software productivity seems to be barely holding its own. Central processing units, random access memories, and mass memories improve their price-performance ratios by orders of magnitude per decade, while software projects continue to grind out production-engineered code at the same old rate of one to two delivered lines of code per man-hour.

Yet, if software is judged by the same standards as hardware, its productivity looks pretty good. One can produce a million copies of Lotus 1-2-3 at least as cheaply as a million copies of the Intel 286. Database management systems that cost \$5 million 20 years ago can now be purchased for \$99.95.

The commodity for which productivity has been slow to increase is custom software. Clearly, if you want to improve your organization's software price-performance, one major principle is "Don't build custom software where mass-produced software will satisfy your needs." However, even with custom software, a great deal is known about how to improve its productivity, and even increasing productivity by a factor of 2 will make a significant difference for most organizations.

This article discusses avenues of improving productivity for both custom

By 1995, a 20 percent improvement in software productivity will be worth \$90 billion worldwide. Clearly, the measures outlined in this article are worth the effort.

and mass-produced software. Its main sections cover the following topics:

- The importance of improving software productivity: some national, international, and organizational trends indicating the significance of improving software productivity.
- Measuring software productivity: some of the pitfalls and paradoxes in defining and measuring software productivity and how best to deal with them.
- Analyzing software productivity: identifying factors that have a strong

productivity influence and those that have relatively little influence, using such concepts as software productivity ranges, the software value chain, and the software productivity opportunity tree.

- Improving software productivity: using the opportunity tree as a framework for describing specific productivity improvement steps and their potential payoffs.
- Software productivity trends and conclusions.

The importance of improving software productivity

The major motivation for improving software productivity is that software costs are large and growing larger. Thus, any percentage savings will be large and growing larger as well. Figure 1 shows recent and projected software cost trends in the United States and worldwide. In 1985, software costs totaled roughly \$11 billion in the US Department of Defense, \$70 billion in the US overall, and \$140 billion worldwide. If present software cost growth rates of approximately 12 percent per year continue, the 1995 figures will be \$36 billion for the DoD, \$225 billion for the US, and \$450 billion worldwide. Thus,

Measuring software productivity

The best definition of the productivity of a process is

$$\text{Productivity} = \frac{\text{Outputs produced by the process}}{\text{Inputs consumed by the process}}$$

Thus, we can improve the productivity of the software process by increasing its outputs, decreasing its inputs, or both. However, this means that we need to provide meaningful definitions of the inputs and outputs of the software process.

Defining inputs. For the software process, providing a meaningful definition of *inputs* is a nontrivial but generally workable problem. Inputs to the software process generally comprise labor, computers, supplies, and other support facilities and equipment. However, one has to be careful which of various classes of items are to be counted as inputs. For example:

- Phases (just software development, or should we include system engineering, software requirements analysis, installation, or postdevelopment support?)
- Activities (to include documentation, project management, facilities management, conversion, training, database administration?)
- Personnel (to include secretaries, computer operators, business managers, contract administrators, line management?)
- Resources (to include facilities, equipment, communications, current versus future dollar payments?)

An organization can usually reach an agreement on which of the above are meaningful as inputs in their organizational context. Frequently, one can use present-value dollars as a uniform scale for various classes of resources.

Defining outputs. The big problem in defining software productivity is defining *outputs*. Here we find a paradox. Most sources say that defining delivered source instructions (DSI) or lines of code as the output of the software process is totally inadequate, and they argue that there are a number of deficiencies in using DSI. However, most organizations doing practical productivity measurement still use DSI as their primary metric.

DSI does have the following deficiencies as a software productivity metric:

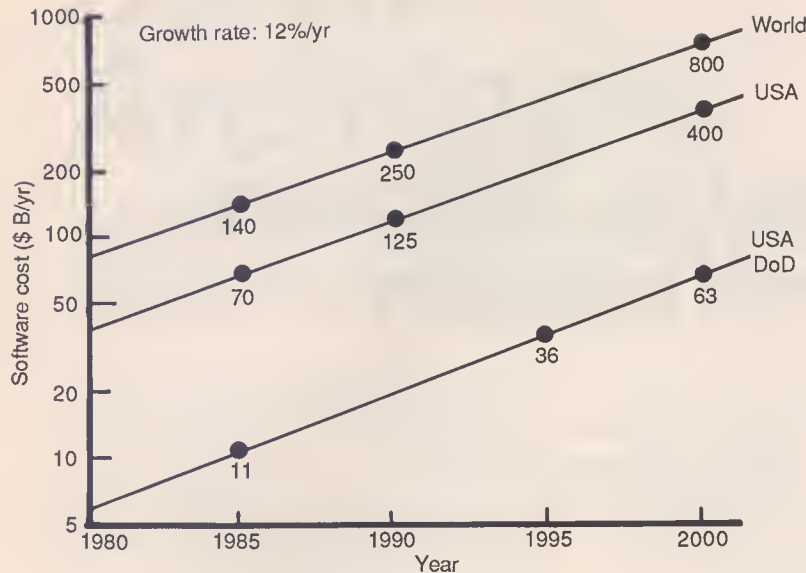


Figure 1. Software cost trends.

even a 20 percent improvement in software productivity would be worth \$45 billion in 1995 for the US and \$90 billion worldwide. Gains of such magnitude are clearly worth a serious effort to achieve.

Software costs are increasing not because people are becoming less productive but because of the continuing increase in demand for software. Figure 2, based on Boehm¹ and a recent TRW-NASA Space Station software study, shows the growth in software demand across five generations of the US manned space flight program, from about 1,500,000 object instructions to support Project Mercury in 1962-63 to about 80,000,000 object instructions to support the Space Station in the early 1990's.

The reasons for this increased demand are basically the same ones encountered by other sectors of the economy as they attempt to increase productivity via automation. The major component of growth in the Space Shuttle software has been the checkout and launch support area, in which NASA automated many functions

to reduce the number of people needed to support each launch—as many as 20,000 in previous manned spaceflight operations. The result has been a significant reduction in required launch support personnel but a significant increase in the required amount of software.

Many organizations have software demand growth curves similar to Figure 2. A large number of organizations simply cannot handle their increased demand within their available personnel and budget constraints, and they are faced with long backlogs of unimplemented information processing systems and software improvements. For example, the US Air Force Standard Information Systems Center has identified a four-year backlog of unstarted projects representing user-validated software needs. This type of backlog serves as a major inhibitor of a software user organization's overall productivity, competitiveness, and morale. Thus, besides cost savings, another major motivation for improving software productivity is to break up these software logjams.

(1) It is too low-level for some purposes, particularly for software cost estimation, where it is often difficult to estimate DSI in advance.

(2) It is too high-level for some purposes because complex instructions or complex combinations of instructions receive the same weight as a sequence of simple assignment statements.

(3) It is not a uniform metric; lines of machine-oriented language (MOL), higher-order language (HOL), and very high level language (VHLL) are given the same weight. For example, completing an application in one man-month and 100 lines of VHLL (100 DSI/MM) should not be considered less productive than doing the same application in two man-months and 500 lines of HOL (250 DSI/MM).

(4) It is hard to define well, particularly in determining whether to count comments, nonexecutable lines of code, reused code, or a "line" as a card image, carriage return, or semicolon. For example, putting a compact Ada program through a pretty printer will frequently triple its number of card images.

(5) It is not necessarily well correlated with value added, in that motivating people to improve productivity in terms of DSI may tempt them to develop a lot of useless lines of code.

(6) It does not reflect any consideration of software quality; "improving productivity" may tempt people to produce faster but sloppier code.

A number of alternatives to DSI have been advanced:

- "Software science" or program information-content metrics
- Program control-flow complexity metrics
- Design complexity metrics
- Program-external metrics, such as number of inputs, outputs, inquiries, files interfaces, or function points, a linear combination of those five quantities²
- Work transaction metrics

Comparing the effectiveness of these productivity metrics to a DSI metric, the following conclusions can be advanced: Each has advantages over DSI in some situations; each has more difficulties than DSI in some situations; each has equivalent difficulties to DSI in relating software achievement units to measures of the software's value added to the user organization.

As an example, let us consider function

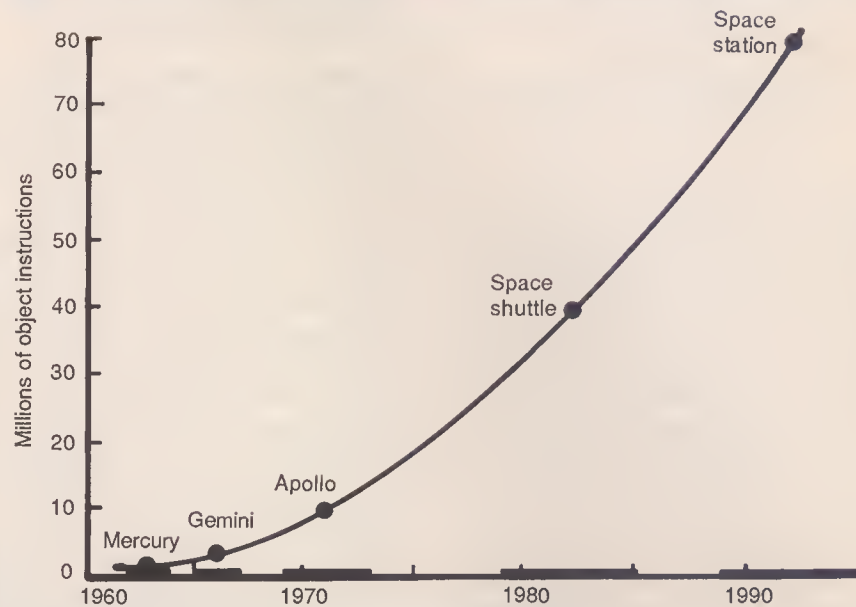


Figure 2. Growth in software demand: US manned spaceflight program.

points, which are defined as

$$\begin{aligned} \text{FPs} = & 4 \times \# \text{Inputs} + 5 \times \# \text{Outputs} + \\ & 4 \times \# \text{Inquiries} + \\ & 10 \times \# \text{Masterfiles} + \\ & 7 \times \# \text{Interfaces}, \end{aligned}$$

where #Inputs means "number of inputs to the program," and so on for the other terms.

Function points offer some strong advantages in addressing problems 1 (too low-level) and 3 (nonuniformity) above. One generally has a better early idea of the number of program inputs, outputs, etc., and the delivered software functionality has the same numeric measure whether the application is implemented in an MOL, HOL, or VHLL. However, function points do not provide any advantage in addressing problems 5 and 6 (value added and quality considerations), and they have more difficulties than DSI with respect to problems 2 and 4 (too high-level and imprecise definition). The software functionality required to transform an input into an output may be very trivial or very extensive. And we still lack a set of well-

rationalized, unexceptionable standard definitions for number of inputs, number of outputs, and other terms that are invariant across designers of the same application. For example, some experiments have shown an order-of-magnitude variation in estimating the number of inputs to an application.

However, function points have been successfully applied in some limited, generally uniform domains such as small-to-medium-sized business applications programs. A number of activities are also under way to provide more standard counting rules and to extend the metric to better cover other software application domains.

Thus, no alternative metrics have demonstrated a clear superiority to DSI. And DSI has several advantages that induce organizations to continue to use DSI as their primary software productivity output metric:

- The DSI metric is relatively easy to define and discuss unambiguously.
- It is easy to measure.

- It is conceptually familiar to software developers.
- It is linked to most familiar cost estimation models and rules of thumb for productivity estimation.
- It provides continuity from many organizations' existing database of project productivity information.

Software productivity-quality interactions. As discussed above, we want to define *productivity* in a way that does not compromise a project's concern with software quality. The interactions between software cost and the various software qualities (reliability, ease of use, ease of modification, portability, efficiency, etc.) are quite complex—as are the interactions between the various qualities themselves. Overall, though, there are two primary situations which create significant interactions between software costs and qualities:

(1) A project can reduce software development costs at the expense of quality but only in ways that increase operational and life-cycle costs.

(2) A project can simultaneously reduce software costs and improve software quality by intelligent and cost-effective use of modern software techniques.

One example of situation 1 was provided by a software project experiment in which several teams were asked to develop a program to perform the same function, but each team was asked to optimize a different objective. Almost uniformly, each team finished first on the objective they were asked to optimize, and fell behind on the other objectives. In particular, the team asked to minimize effort finished with the smallest effort to complete the program, but also finished last in program clarity, second to last in program size and required storage, and third to last in output clarity.

Another example is provided by the Cocomo database of 63 development projects and 24 evolution or maintenance projects.¹ This analysis showed that if the effects of other factors such as personnel, use of tools, and modern programming practices were held constant, then the cost to develop reliability-critical software was almost twice the cost of developing minimally reliable software. However, the trend was reversed in the maintenance projects; low-reliability software required considerably more budget to maintain than high-reliability software. Thus, there is a "value of quality," which makes it generally undesirable in the long run to

reduce development cost at the expense of quality.

Certainly, though, if we want better software quality at a reasonable cost, we are not going to hold constant our use of tools, modern programming practices, and better people. This leads to situation 2, in which many organizations have been able to achieve simultaneous improvements in both software quality and productivity. For example, the extensive Guide, Inc., survey of about 800 user installations found that the four most strongly experienced effects of using modern programming practices were code quality, early error detection, programmer productivity, and maintenance time or cost. Also, the Cocomo life-cycle data analysis indicated that the use of modern programming practices had a strong positive impact on development productivity but an even stronger positive impact on maintenance productivity.

However, getting the right mix of the various qualities (reliability, efficiency, ease of use, ease of change, etc.) can be a very complex job. Several studies have explored these qualities and their interactions. Also, several new approaches have had some success in providing methods for reconciling and managing multiple quality objectives, such as Gilb's design by objectives and the Goals approach (Boehm,¹ Chapter 3). For pointers to additional information on these and other topics covered in this article, see the "Further Reading" box on pp. 54-55.

Metrics: The current bottom line. The current bottom line for most organizations is that delivered source instructions per project man-month (DSI/MM) is a more practical productivity metric than the currently available alternatives. To use DSI/MM effectively, though, it is important to establish a number of measurement standards and interpretation guidelines, including

- objective, well-understood counting rules defining which project-related man-months are included in MM;
- objective, well-understood counting rules for source instructions;
- a definition of *delivered* in terms of compliance with a set of software quality standards;
- definition and tracking of the language level and extent of reuse of source instructions, along with interpretation guidelines encouraging

the use of VHLLs, HOLs, and reused software.

Examples of such definitions are given by Boehm¹ and by Jones.²

In addition, because new metrics such as function points have been successful in some areas, many organizations are also experimenting with their use, refinement, and extension to other areas.

Analyzing software productivity

We can consider two primary ways of analyzing software productivity:

(1) The "black-box" or influence-function approach, which performs comparative analyses on the overall results of a number of entire software projects, and which tries to characterize the overall effect on software productivity of such factors as team objectives, methodological approach, hardware constraints, turnaround time, or personnel experience and capability.

(2) The "glass-box" or cost-distribution approach, which analyzes one or more software projects to compare their internal distribution between such costs as labor and capital, code and documentation, development and maintenance, and other cost distributions by phase or activity.

Here, we will concentrate on two representative approaches: the black-box productivity range and the glass-box value-chain.

Software productivity ranges. Most software cost estimation models incorporate a number of software cost driver factors: attributes of a software project or product that affect the project's productivity in (appropriately defined) DSI/MM. A significant feature of some of these models is the productivity range for a software cost driver: the relative multiplicative amount by which that cost driver can influence the software project cost estimated by the model. An example of a set of recently updated productivity ranges for the Cocomo models is shown in Figure 3.

These productivity ranges show the relative leverage of each factor on one's ability to reduce the amount of effort required to develop a software product. For example, assuming all the other factors are held constant, developing a software product in an unfamiliar programming language will

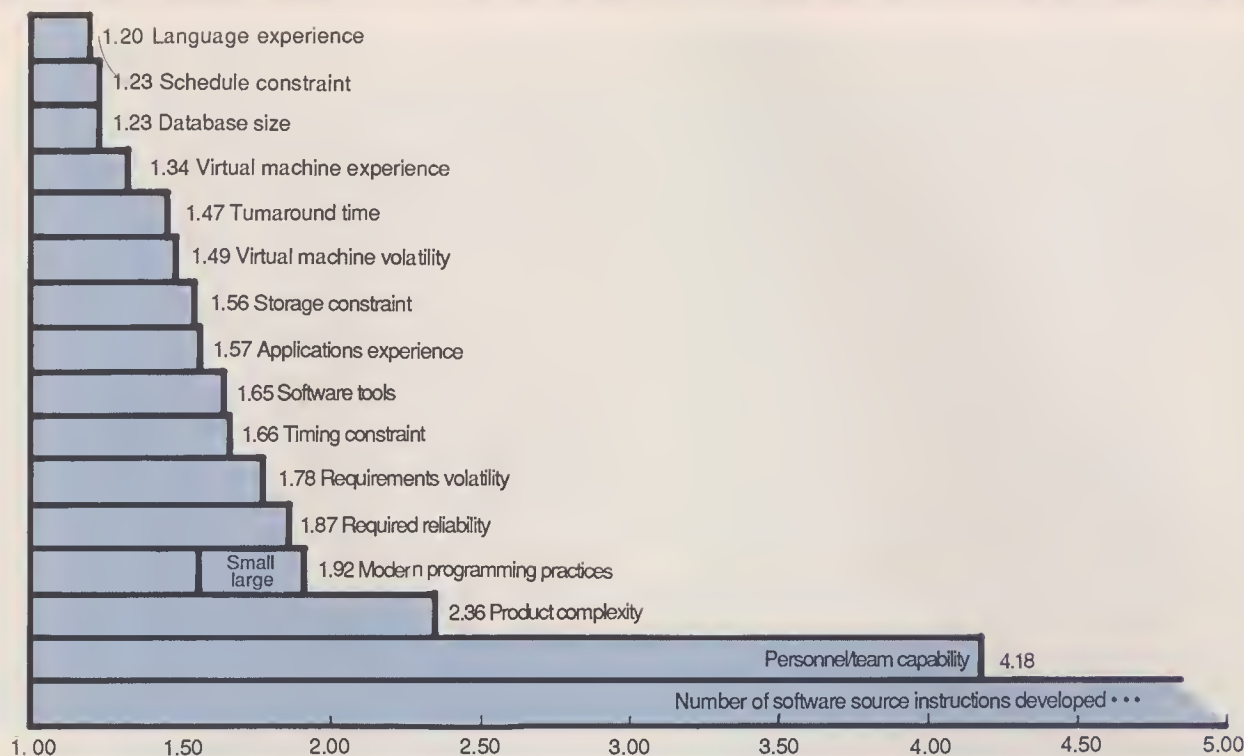


Figure 3. Cocomo software life-cycle productivity ranges, 1985.

typically require about 20 percent more man-months than using a very familiar language. Similarly, developing a product with a mediocre (15th-percentile) team of people will typically require over four times as many man-months as with a 90th-percentile team of people. The open-ended bar at the bottom of Figure 3 indicates that the number of man-months required to develop a software product increases without bound as one increases the number of instructions developed.

Some initial top-level implications of the productivity ranges are summarized as follows; more detailed implications will be discussed in "Improving software productivity" later in this article.

- *Number of source instructions.* The most significant influence on software costs is the number of source instructions one chooses to program. This leads to cost reduction strategies involving the use of fourth-generation languages or reusable components to reduce the number of source instructions developed, the use of prototyping and other requirements anal-

ysis techniques to ensure that unnecessary functions are not developed, and the use of already-developed software products.

- *Management of people.* The next most significant influence by far is that of the selection, motivation, and management of the people involved in the software process. In particular, employing the best people possible is usually a bargain, because the productivity range for people usually is much wider than the range of people's salaries.

- *Fixed features of the product.* Some of the factors, such as product complexity, required reliability, and database size, are largely fixed features of the software product and not management controllables. Even here, though, appreciable savings can be achieved by reducing unnecessary complexity and by focusing on appropriate life-cycle cost-reliability trade-offs as discussed in the preceding section.

- *Other, management-controllable factors.* The other cost driver factors are generally management controllables:

requirements volatility, hardware speed and storage constraints, use of software tools and modern programming practices, and so on can be directly factored into a productivity improvement effort.

Some primary productivity improvement strategies involving these cost driver variables are described later. See Boehm,¹ Chapter 33, for a discussion of each cost driver and Boehm et al.³ for an example of their successful application to an integrated software productivity improvement program.

The software product value chain. The value chain, developed by Porter and his associates at the Harvard Business School,⁴ is a useful method of understanding and controlling the costs involved in a wide variety of organizational enterprises. It identifies a canonical set of cost sources or value activities, representing the basic activities an organization can choose from to create added value for its products. Figure 4 shows a value chain for software development representative of

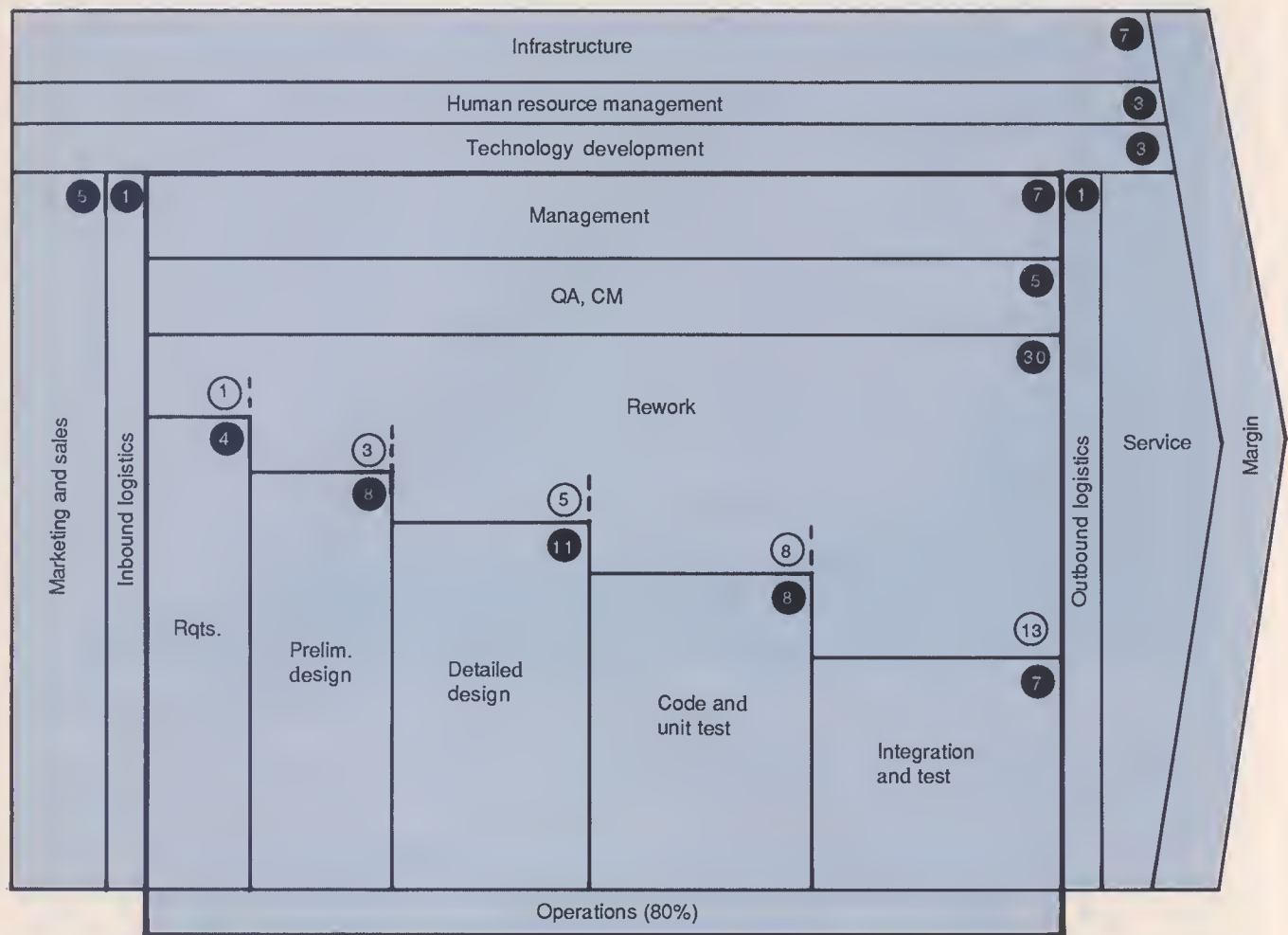


Figure 4. Typical software development value chain.

experience at TRW. Definitions and explanations of the component value activities are given by Porter.⁴

For software the largest single value chain element is Operations, which covers activities associated with transforming inputs into the final product form and typically involves roughly four-fifths of the total development outlay. In such a case, the value chain analysis involves breaking up a large component into constituent activities. Figure 4 shows such a breakup of Operations into management (7 percent), quality assurance and configuration management (5 percent), and the distribution of technical effort among the

various development phases. This phase breakdown also covers the cost sources due to rework. Thus, for example, of the 20 percent overall cost of the technical effort during the integration and test phase, 13 percent is devoted to activities required to rework deficiencies in or reorientations of the requirements, design, code, or documentation; the other 7 percent represents the amount of effort required to run tests, perform integration functions, and complete documentation even if no problems were detected in the process.

For simplicity, the service and margin components of the value chain have not

been assigned numerical values. "Margin" basically represents profits; "service" represents postdevelopment software support activities, often called "maintenance" but more properly called "evolution." Evolution costs are typically 70 percent of software life-cycle costs, but since some initial analyses have indicated that the detailed value chain distribution for evolution costs is not markedly different from the distribution of development costs in Figure 4, we will use Figure 4 to represent the distribution of software life-cycle costs.

The primary implication of the software development value chain is that the Oper-

ations component is the key to significant improvements. Not only is it the major source of software costs, but also most of the remaining components such as human resources will scale down in a manner roughly proportional to the scaling down of Operations cost.

Another major characteristic of the value chain is that virtually all of the components are still highly labor-intensive. Thus, there are significant opportunities for providing automated aids to make these activities more efficient and capital-intensive. Further, it implies that human resource and management activities aimed at *getting the best from people* have much higher leverage than their 3 percent and 7 percent investment levels indicate.

The breakdown of the Operations component indicates that the leading strategies for cost savings in software development involve

- *making individual steps more efficient* via such capabilities as automated aids to software design analysis or testing;
- *eliminating steps* via such capabilities as automatic programming or automatic quality assurance;
- *eliminating rework* via early error detection or via such capabilities as rapid prototyping to avoid later requirements rework.

In addition, further major cost savings can be achieved by reducing the total number of elementary operations steps by developing products requiring the creation of fewer lines of code. This has the effect of reducing the overall size of the value chain itself. This source of savings breaks down into two main options:

- *building simpler products* by applying more insight to front-end activities such as prototyping or risk management;
- *reusing software components* via such capabilities as fourth-generation languages or component libraries.

The software productivity improvement opportunity tree. This breakdown of the major sources of software cost savings leads to the software productivity improvement opportunity tree shown in Figure 5. This hierarchical breakdown helps us to understand how to fit the various attractive productivity options into an overall integrated software productivity improvement strategy. The next section will discuss each of these major options in turn.

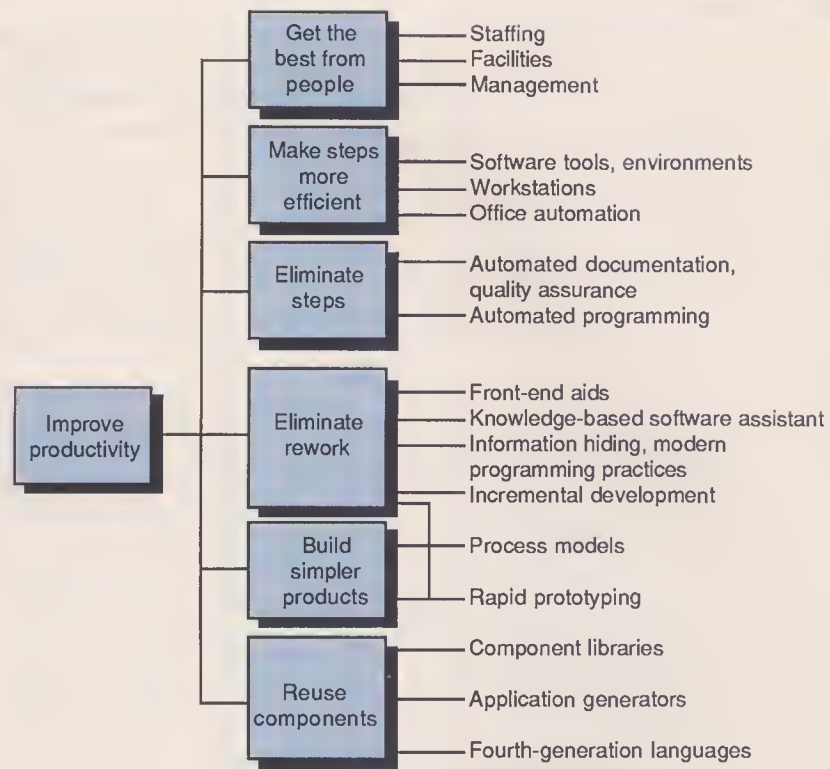


Figure 5. Software productivity improvement opportunity tree.

Improving software productivity

Following the organization of the software productivity opportunity tree, we will cover the following primary options for improving software productivity: (1) getting the best from people, (2) making steps more efficient, (3) eliminating steps, (4) eliminating rework, (5) building simpler products, and (6) reusing components.

Getting the best from people. As indicated in the opportunity tree, there are three primary options available for getting the best from people: staffing, facilities, and management.

Staffing. The productivity ranges in Figure 3 show a factor of 4.18 in productivity difference due to personnel/team capability and a combined factor of 2.52 for relative experience with the applica-

tions area, computer system or virtual machine, and programming language. Similar ranges have been determined by other studies such as the IBM productivity analysis (see Walston and Felix⁵).

Thus, if you want to increase your project's or organization's software productivity, one of the biggest leverage actions you have at your disposal is to get the best people working for your project or organization and the mediocre people working for someone else. It is worth making a significant effort to get this to happen. But it is remarkable how frequently managers are passive about key staffing decisions and how frequently they go in the opposite direction, saying things like

"I can't afford those high-salary people."

"I can't take a risk on somebody so expensive."

"I can't hire your superstar until your project gets its funds, even

though that's only a month away." "Joe has all these unassigned people charging to standby, and I have to help him out."

"I can't wait. I need somebody to show some progress on this task by next week."

Sometimes the latter two situations require you to respond, but you can generally make it a temporary rather than a permanent commitment.

The other, equally important, side of the staffing coin involves committing yourself to phase out misfits. No matter how carefully you select the members of your software team, inevitably you will find some people who do not contribute anywhere near their fair share to the team's objectives, even after several attempts to find or prepare a suitable role for them on the team. In such a situation, you will be tempted to postpone dealing with the problem, to profess not to notice it, to smooth it over with words, or to ask the other team members to do extra tasks. This may be the easy way out in the short run, but invariably it produces unhealthy results in the long run.

Phasing people out isn't easy. But if you devote enough time, thought, and sympathy to the problem, you can often create a situation in which the phaseout becomes a positive rather than a negative experience, and the person concerned finds a new line of work which suits him or her much better than a group-oriented software project. If this doesn't work, and you are left with a definite misfit, don't back away from the problem. Get rid of the misfit as quickly as possible.

Facilities. Given that software development and evolution are extremely labor-intensive activities, a great deal of productivity leverage can be gained by making software production a more capital-intensive activity. Typically, capital investment per software worker has been little different from the \$2000-\$3000 per person typical of office workers in general. However, a number of organizations such as Xerox, TRW, IBM, and Bell Laboratories have indicated that significantly higher investments (in the \$10,000-\$30,000 per person range) have been more than recaptured in improved software productivity.

Providing software personnel with private offices is another cost-effective measure, leading to productivity gains of roughly 11 percent at IBM-Santa Teresa (see Jones²) and 8 percent at TRW

(Boehm et al.³). Similar results on the payoffs of capital investments in better facilities and support capabilities have been reported in other studies (see "Further Reading").

Management. Poor management can decrease software productivity more rapidly than any other factor. Here are some examples of the major classes of management activities that most frequently contribute to losses in productivity:

- **Poor planning.** An example of poor planning was a project with very vague test plans. When the 20-person test team came on board, they found no test data, test drivers, test facilities, test strategies and procedures, or test readiness standards for the developers' code. As a result, the project incurred a 30 percent overrun in cost and a 40 percent overrun in schedule.

- **Poor skill mix.** Poor skill mix is often a result of the Peter Principle: "In a hierarchy, every employee tends to rise to his level of incompetence." The most common realization of the Peter Principle in software engineering is the practice of "advancing" good programmers by promoting them into management. Sometimes this works well, but overall it produces more mismatches, frustration, and damaged careers in software engineering than in other fields. This point has been realized by a number of organizations, which have instituted dual or multiple career paths culminating in "super-programmer" or "superanalyst" as well as "supermanager."

- **Premature staffing.** An example of premature staffing is the following quotation from a small-project manager: "At an early stage in the design, I was made the project manager and given three trainees to help out on the project. My biggest mistake was to burn up half of my time and the other senior designer's time trying to keep the trainees busy. As a result, we left big holes in the design which killed us in the end." A related source of decreased productivity is the attempt to speed up a project by adding more people, in contradiction to Brooks's law: "Adding more people to a late software project will make it later."

- **Premature coding.** An example of premature coding is the WISCA syndrome, where WISCA stands for "Why

isn't Sam coding anything?" A counterpart is the statement, "We'd better hurry up and start coding, because we're going to have a lot of debugging to do." The most important management property of an efficient multiperson software development is the achievement of a set of thorough, validated, and stable module interface specifications, which allow the developers to operate in parallel without being swamped by interpersonal communications overhead. As early as 1961, software managers were realizing that "every sheet of accurate interface definition is, quite literally, worth its weight in gold."

- **Poor reward structure.** An example of poor reward structure is the organization which gives its top performers six percent raises and its mediocre performers five percent raises. Eventually, the good people get frustrated and leave. A great deal can be done by creative application of other rewards, such as special bonuses, grade-level promotions, travel and special courses, and recognition programs for top performers.

Making steps more efficient. The value chain in Figure 4 provides a basic set of insights on the relative productivity leverage involved in eliminating or improving the efficiency of the various steps in the software process. For example, since the process of performing code and unit test functions consumes only eight percent of the software life-cycle dollar, the productivity impact of tools to eliminate code and unit test or to make it more efficient will not exceed eight percent (unless the tools also eliminate other classes of effort, such as rework in later phases).

The primary leverage factor in making the existing software process steps more efficient is the use of software tools to automate the current repetitive and labor-intensive portions of each step.

Experience to date suggests that software tools are much more effective if they are part of an integrated project support environment (IPSE). The primary features that distinguish an IPSE from an ad hoc collection of tools are

- a set of common assumptions about the software process model being supported by the tools (or, more strongly, a particular software development method being supported by the tools);
- an integrated project master database or persistent object base serving as a unified repository of the technical

and management entities created during the software process, along with their various versions, attributes, and relationships;

- support of the entire range of users and activities involved in the software project, not just of programmers developing code;
- a unified user interface providing easy and natural ways for various classes of project personnel (expert programmers, novice librarians, secretaries, managers, planning and control personnel, etc.) to draw on the tools in the IPSE;
- a critical-mass ensemble of tools, covering significant portions of software project activities;
- a computer communication architecture facilitating user access to data and resources in the IPSE.

Eliminating steps. A good many automated aids go beyond simply making steps more efficient, to the point of fully eliminating previous manual steps. If we compare software development today with its counterpart in the 1950's, we see that assemblers and compilers are excellent examples of ways to vastly improve productivity by eliminating steps. More recent examples are process construction systems, software standards checkers and other quality assurance functions, and requirements and design consistency checkers.

More ambitious efforts to eliminate steps involve the automation of the entire programming process by providing capabilities which operate directly on a set of software specifications to automatically generate computer programs. There are two major branches to this approach: domain-specific and domain-independent automatic programming.

The domain-specific approach gains advantages by capitalizing on domain knowledge in transforming specifications into programs and in constraining the universe of programming discourse to a relatively smaller domain. In the limit, one reaches the boundary with fourth-generation languages such as Visicalc, which are excellent automatic programming systems within a very narrow domain and relatively ineffective outside that domain.

The domain-independent approach offers a much broader payoff in the long run but has more difficulty in achieving efficient implementations of larger-scale programs.



Eliminating rework. The strongest opportunity identified by the value chain analysis in Figure 4 is the 30 percent productivity leverage available through eliminating rework. Actually, the leverage factor is probably more like 50 percent over the life cycle, since most of the sources of rework savings (e.g., modern programming practices and rapid prototyping) will reduce the incidence of current postdevelopment software modifications (e.g., to fix residual errors or to finally get the requirements right) as well as making the modifications more efficient.

The major rework opportunity areas identified in the opportunity tree in Figure 5 are front-end aids; knowledge-based software assistants; information hiding and modern programming practices, incremental development, improved process models, and rapid prototyping. (In addition, reusing components can significantly reduce rework.)

Front-end aids. Software computer-aided design and requirements analysis tools can eliminate a great deal of rework through better visualization of software requirements and design specification, more formal and unambiguous specifications, automated consistency and completeness checking, and automated traceability of requirements to design. Probably the most extensive of these systems is the Distributed Computing Design System, which includes a system specification language, a software requirements specification language, a distributed-system design language, and a module description language. A number of commercial front-end aids are also available, such as ISDOS/PSL-PSA, SADT, CASE, Excelsator, IDE, Cadre, and Ada Graph. Some complementary front-end aids include rapid simulation aids such as RSA and executable specification aids such as Paisley.

Knowledge-based software assistants. In many application areas, the artificial intelligence community is finding that total automation of knowledge-intensive functions falls in the "currently too hard" category but that combinations of conventional and AI techniques may be used to provide useful automated assistance to human experts in performing complex tasks. This is the primary moti-

vation for the knowledge-based software assistant (KBSA) concept, as described by Green et al.⁶

The primary benefit of a KBSA will be the elimination of much of the rework currently experienced on software projects due to the belated appreciation that a previous programming or project decision was inappropriate. A number of prototype KBSAs are currently under development in such areas as acquisition management, configuration management, problem report tracking, algorithm selection, data structuring, choice of reusable components, and project planning and control.

Information hiding and other modern programming practices. In general, modern programming practices (MPPs) such as early verification and validation, modular design, top-down development, structured programming, walk-throughs or inspections, and software quality standards achieve their productivity leverage through avoidance of rework. As indicated in Figure 3, MPPs provide a productivity range of 1.51 during development and up to 1.92 for the life cycle of a large software product.

A particularly powerful technique for eliminating rework is the information-hiding approach developed by Parnas and applied in the US Navy A-7 project (Parnas, Clements, and Weiss.⁷ This approach minimizes rework by hiding implementation decisions within modules; thus minimizing the ripple effects usually encountered when software implementation decisions need to be changed. The information-hiding approach can be particularly effective in eliminating rework during software evolution, by identifying the portions of the software most likely to undergo change (characteristics of workstations, input data formats, etc.) and hiding these sources of evolutionary change within modules.

As an example, the current requirements may specify that a particular user workstation or terminal is to be used. By also identifying in the requirements the terminal characteristics most likely to change (line width, character set, access protocols, etc.), the designers can hide these details of the terminal inside a terminal-handler module, thus isolating the remainder of the software from the usual ripple effects accompanying a change in the terminal characteristics.

This approach revolutionizes the concept of a requirements specification. Rather than being just a snapshot of a sys-

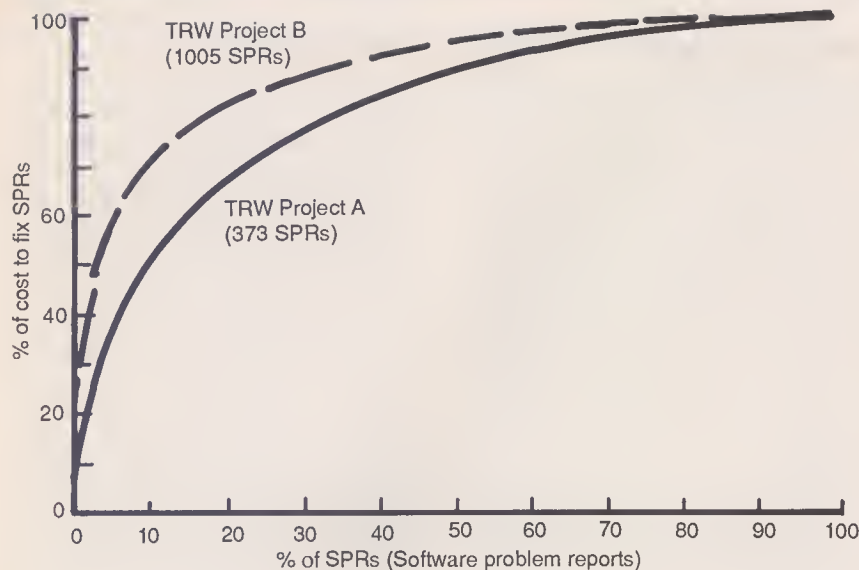


Figure 6. Rework costs are concentrated in a few high-risk items.

tem's software requirements at a single point in time, the requirements specification must also identify the most likely requirements evolution paths the system will experience. This also means that a design validation activity should address not just traceability to the current requirements snapshot but also how well the design accommodates the expected directions of change.

Modern programming practices and Ada. A major initiative to embody modern programming practices and information hiding concepts into standard programming practice has been the US Department of Defense's development of the programming language Ada. Ada has constructs such as packages that support modularity, information hiding, and reuse; strong typing that avoids rework due to common programming errors; structured programming constructs; and a number of other advanced features addressing such issues as concurrency, exception handling, and generic programs. Getting all of these features to work together has strained the state of the art in compiler development, but currently a number of effective Ada compilers are available.

Assessing the likely productivity impact of Ada is difficult because the Ada concept is also intended to include an overall programming and project environment and because most Ada capabilities are not yet fully mature. However, several studies have estimated the comparative life-cycle productivity of an Ada project and a conventional HOL project as a function of time, using as parameters the cost driver variables and productivity ranges of models such as Cocomo. The typical result of these studies has indicated an initial added cost of 12-30 percent for initial uses of Ada, a breakeven point in the 1988-89 time frame, and a long-range savings of 40-50 percent for a fully mature Ada support environment and development staff.

Improved process models. The leading current model of the software process, the waterfall model,¹ tends to focus a software project toward the production of a series of documents (system specification, software requirements specification, top-level design document, detailed design document). When used in concert with thorough front-end validation activities, the waterfall approach is very effective in reducing rework. But, frequently, the document-driven interpretation of the

waterfall model pushes a project toward more rapid production of documents rather than toward thinking through critical issues. For example, a recently proposed government software progress reporting scheme focuses on the number of unresolved elements in the software requirements and design specifications. If a project manager wants to show rapid progress, he is actually tempted to work on resolving the easy elements rather than the hard ones or to complete the document quickly by putting in arbitrary rather than well-reasoned specifications.

An important point in this regard is that rework instances tend to follow a Pareto distribution: 80 percent of the rework costs typically result from 20 percent of the problems. Figure 6 shows some typical distributions from recent TRW software projects; similar trends have been indicated in other studies. The major implication of this distribution is that software verification and validation activities should focus on identifying and eliminating the specific high-risk problems to be encountered by a software project, rather than spreading the early problem elimination effort uniformly across trivial and severe problems. Even more strongly, this implies that a risk-driven approach to the software life cycle such as the spiral model (see Boehm⁸) is preferable to a more document-driven model such as the traditional waterfall model. The spiral model organizes the software development process into a sequence of increasingly detailed definition cycles. The amount of emphasis in each cycle on documentation, simulation, prototyping, or other definition activities is determined by the relative risk of not resolving key definition issues. Thus, the spiral model focuses effort on identification and early resolution on the 20 percent of the problems that will otherwise account for 80 percent of the rework costs.

Rapid prototyping. One of the major sources of rework found in the data represented by Figure 4 were portions of a software specification based on poorly understood mission or user interface requirements. A primary example is the user who says, "I can't tell you exactly what I want, but I'll know it when I see it." A number of rapid prototyping aids have become available to improve this situation. A good many are based on the interpretive-execution capabilities of advanced artificial intelligence environments such as Interlisp. Others are based

on two-phase interactive-graphics composition and execution capabilities using conventional HOLs. Other rapid prototyping systems provide risk reduction capabilities for rapid assessment of real-time performance issues or distributed data processing issues.

Building simpler products. As indicated in Figure 3, the largest productivity range available to the software developer comes from the number of instructions one chooses to develop. There are two primary options here: one is building simpler products; the other is reusing software components.

Besides their contribution to eliminating rework, the last two approaches involving rapid prototyping and improved software process models can also be very effective in improving bottom-line productivity by building simpler products to eliminate software gold plating: extra software that not only consumes extra effort but also reduces the conceptual integrity of the product.

For example, a recent seven-project experiment comparing a specification-oriented approach and a prototyping-oriented approach to the development of small-user-intensive application software products (see Boehm, Gray, and Seewaldt⁹) is illustrated in Figure 7. The experiment found primarily that

- on the average ($\bar{P}$ vs $\bar{S}$ in Figure 7), both approaches resulted in roughly equivalent productivity in delivered source instructions per man-hour (DSI/MH);
- the prototyping projects developed products with roughly equivalent performance but requiring roughly 40 percent fewer DSI and 40 percent fewer man-hours;
- the specifying projects had less difficulty in debugging and integration due to their development of good interface specifications.

The final point indicates that prototypes are not a panacea for all problems and that specifications are still very important. However, one of the telling insights in this experiment was the comment of one of the participants using the specification approach: "Words are cheap." During the specification phase, it is all too easy to add gold-plating functions to the product specification, without a good understanding of their effect on the product's conceptual integrity or the project's required effort. As Heckel¹⁰ writes:

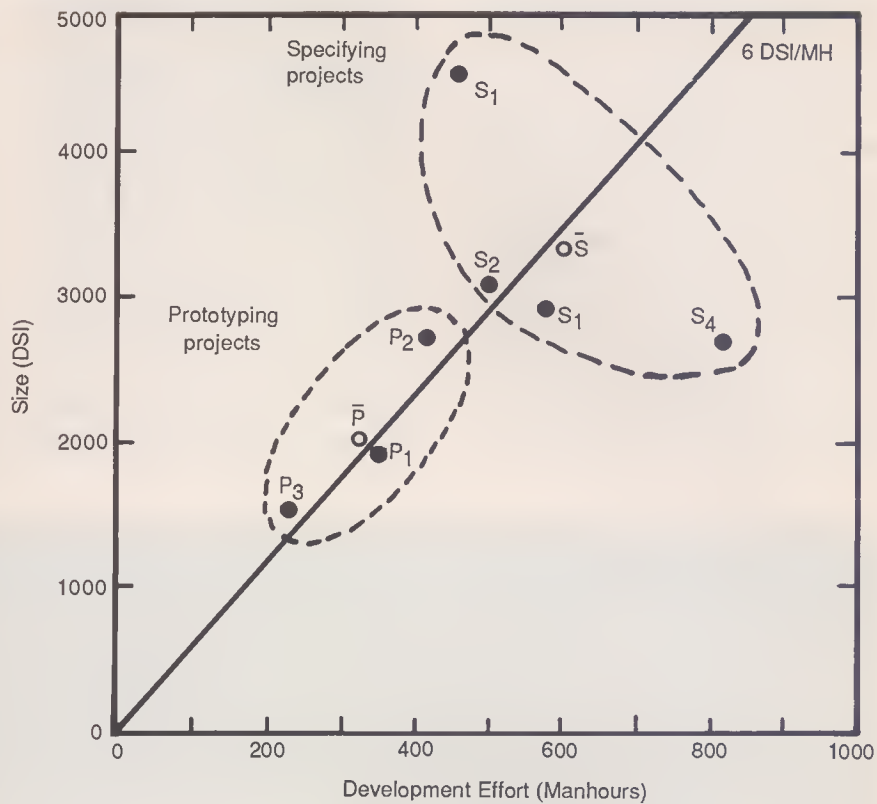


Figure 7. Prototyping versus specifying experiment: size and effort comparisons.

Most programmers . . . defend their use of a software feature by saying, "You don't have to use it if you don't want to, so what harm can it do?" It can do a great deal of harm. The user might spend time trying to understand the feature, only to decide it isn't needed, or he may accidentally use the feature and not know what has happened or how to get out of the mistake. If a feature is inconsistent with the rest of the user interface, the user might draw false conclusions about the other commands. The feature must be documented, which makes the user's manual thicker. The cumulative effect of such features is to overwhelm the user and obscure communication with your program. . . .

A further discussion of typical sources of software gold plating and an approach for evaluating potential gold-plating features is provided by Boehm,¹ Chapter 11.

Some of the newer software process models stimulate the development of simpler products. One of the difficulties of the

traditional waterfall model is that its specification-driven approach can frequently lead one along the "Words are cheap" road toward gold-plated products, as discussed above. A frequent experience in the specification of large systems is that users with little feel for a computer system will overspecify on functionality and performance just to make sure the system will include what they need.

The evolutionary development model (see McCracken and Jackson¹¹) emphasizes the use of prototyping capabilities to converge on the necessary or high-leverage software product features essential to the user's mission. The related transformational model (see Balzer, Cheatham, and Green¹²) shortcuts the problem by providing (where available) a direct transformation from specification to executing code, thus supporting both a specification-based and an evolutionary-development approach. The spiral model discussed

above addresses the gold-plating problem by focusing on a continuing determination of users' mission objectives and a continuing cost-benefit analysis of candidate software product features in terms of their contribution to mission objectives.

Reusing components. Another key to improving productivity by writing less code is the reuse of existing software components. The simplest approach in this direction involves the development and use of libraries of software components.* A great deal of progress has been made in this direction, particularly in such areas as

*There is a good deal of productivity leverage in reusing software specifications, designs, and plans, as well as code.

mathematical and statistical routines and operating-system-related utilities. Further progress is possible via similar capabilities in user application areas. For example, Raytheon's library system of reusable business application components has achieved typical figures of 60 percent reusable code for new applications, and typical cost savings of 10 percent in the design phase, 50 percent in the code and test phase, and 60 percent in the maintenance phase. Toshiba's system of reusable components for industrial process control has resulted in typical productivity rates of over 2000 source instructions per man-month for high-quality industrial software products.

At this level of sophistication, such systems would better be called application

generators, rather than component libraries, because they have addressed several system-oriented component compatibility issues such as component interface conventions, data structuring, and program control and error handling conventions. Similar characteristics have made Unix a strong foundation for developing application generators.

One can proceed even further in this direction to create a very high level language or fourth-generation language (4GL) by adding a language for specifying desired applications and a set of capabilities for interpreting user specifications, configuring the appropriate set of components, and executing the resulting program. Currently, the most fertile areas for 4GLs are spreadsheet calculators (Visicalc,

Further reading

Each of the topics in this article involves a fascinating complex of ideas and capabilities that could only be summarized. Listed below are additional articles and books that amplify some of its specific points or cover one of its main topics in more detail.

Software cost drivers and trends

Boehm, B.W., "Understanding and Controlling Software Costs," *Proc. IFIP Congress 1986*, North Holland, 1986, pp. 703-714. An updated version to appear in *IEEE Trans. Software Engineering*, 1987.

Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

Brooks, F.P., "No Silver Bullet—Essence and Accidents of Software Engineering," *Proc. IFIP Congress 1986*, North Holland, 1986, pp. 1069-1076. Also in *Computer*, Apr. 1987, pp. 10-19.

DeMarco, T., *Controlling Software Projects*, Yourdon, New York, 1982.

Jones, T.C., "Demographic and Technical Trends in the Computing Industry," Software Productivity Research, Inc., July 1983.

Jones, T.C., *Programming Productivity*, McGraw-Hill, New York, 1986.

Walston, C.E., and C.P. Felix, "A Method of Programming Measurement and Estimation," *IBM Systems J.*, Vol. 16, No. 1, 1977, pp. 54-73.

Wegner, P., "Capital-Intensive Software Technology," *IEEE Software*, Vol. 1, No. 3, July 1984, pp. 7-45.

Software productivity and quality metrics

Basili, V., *Tutorial on Models and Metrics for Software Management and Engineering*,

Computer Society Press, Los Alamitos, Calif., 1980.

Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

Boehm, B.W., et al., *Characteristics of Software Quality*, North Holland, 1978.

Bowen, T.P., G.B. Wigle, and J.T. Tsai, "Specification of Software Quality Attributes," RADC-TR-85-37 (3 vol.), Feb. 1985.

Conte, S.D., H. Dunsmore, and V. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings, 1986.

DeMarco, T., *Controlling Software Projects*, Yourdon, New York, 1982.

Frewin, E., et al., "Quality Measurement and Modeling—State of the Art Report," ESPRIT Report REQUEST/STC-gdf/001/51/QL-RP/00.7, July, 1985.

Gilb, T., *Principles of Software Engineering Management*, Addison-Wesley, 1987.

Grady, R., and D. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.

"GUIDE Survey of New Programming Technologies," *GUIDE 1979 Proceedings*, Guide, Inc., 1979, pp. 306-308.

Halstead, M.H., *Elements of Software Science*, Elsevier, New York, 1977.

Jones, T.C., *Programming Productivity*, McGraw-Hill, New York, 1986.

McCabe, T.J., "A Complexity Measure," *IEEE Trans. Software Engineering*, Dec. 1976, pp. 308-320.

Thadhani, A.J., "Factors Affecting Programmer Productivity During Application Development," *IBM Systems J.*, Vol. 23, Nov. 1984, pp. 19-35.

Weinberg, G.M., and E.L. Schulman, "Goals and Performance in Computer Programming," *Human Factors*, Vol. 16, No. 1, 1974, pp. 70-77.

Getting the best from people

Brooks, F.P., *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1975.

Couger, J.D., and R.A. Zawacki, *Motivating and Managing Computer Personnel*, John Wiley, New York, 1980.

Curtis, B., *Human Factors in Software Development*, IEEE Cat. No. EHO 185-9, 1981.

DeMarco, T., *Controlling Software Projects*, Yourdon, New York, 1982.

DeMarco, T., and T. Lister, "Programmer Performance and the Effects of the Workplace," *Proc. Eighth Int'l Conf. Software Engineering*, Aug. 1985, pp. 268-272.

Hosier, W.A., "Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming," *IRE Trans. Engineering Management*, June 1961, pp. 99-115.

Manley, J.H., "Software Engineering Provisioning Process," *Proc. Eighth Int'l Conf. Software Engineering*, Aug. 1985, pp. 273-284.

Metzger, P.J., *Managing a Programming Project*, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1981.

Weinberg, G.M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.

Multiplan, 1-2-3, etc.) and small-business systems typically featuring a DBMS, report generator, database query language, and graphics package (Nomad, Ramis, Focus, ADF, DBase II, etc.).

Some 4GL advocates promise factors of 10 to 100 improvement in productivity from the use of 4GLs. Are such factors achievable?

The best experimental evidence on the productivity leverage of 4GLs is provided by a six-project experiment comparing the use of a third-generation programming language (Cobol) and a fourth-generation language (Focus) on a mix of small-business-application projects involving both experts and beginners developing both simple and complex applications (see Harel and McLean¹³). Its primary find-

ings, illustrated in Figure 8, are summarized as follows:

- On an overall average (the $\bar{C}$ and the $\bar{F}$ in Figure 8), the fourth-generation approach produced equivalent products to the third-generation approach, with about 60 percent fewer DSI and 60 percent fewer man-hours (again with roughly equivalent productivity in DSI/MH).
- From project to project, there was a significant variation in the ratio of third-generation to fourth-generation DSI (0.9:1 to 27:1), man-hours (1.5:1 to 8:1), and DSI/MH (0.5:1 to 5:1).

Although the average Cobol effort was 2.5 times higher than the average Focus effort for the same application, the effect

is far from uniform across a spectrum of applications. Thus, it is difficult to predict the 4GL productivity gain for any particular application.

Guimaraes¹⁴ provides further evidence from a survey of 43 organizations that 4GLs reduce personnel costs, reduce user frustration, and more quickly satisfy user information needs within their domain of applicability. On the other hand, the survey found 4GLs extremely inefficient of computer resources and difficult to interface with conventional applications programs. Some major disasters have occurred in attempting to apply purely 4GL solutions to large, high-performance applications such as the New Jersey motor vehicle registration system.¹⁵

Overall, though, 4GLs offer an

Weinberg, G.M., and E.L. Schulman, "Goals and Performance in Computer Programming," *Human Factors*, Vol. 16, No. 1, 1974, pp. 70-77.

Yourdon, E., *Managing the Structured Techniques*, 3rd ed., Yourdon Press, 1987.

Making steps more efficient

Alford, M.W., "SREM at the Age of Eight: The Distributed Computing Design System," *Computer*, Vol. 18, No. 4, Apr. 1985.

Boehm, B.W., et al., "A Software Development Environment for Improving Productivity," *Computer*, Vol. 17, No. 6, June 1984, pp. 30-44.

Green, C.C., et al., "Report on a Knowledge-Based Software Assistant," USAF/RADC Report RADC-TR-195, Aug. 1983.

Henderson, P., ed., *Proc. Second Conf. Practical Software Development Environments*, ACM SIGPLAN Notices, Dec. 1986.

Houghton, R.C., "Software Development Tools," National Bureau of Standards, NBS Special Report 500-88, 1982.

Sommerville, I., ed., *Software Engineering Environments*, Peter Peregrinus, Ltd., 1987.

Wegner, P., "Capital-Intensive Software Technology," *IEEE Software*, Vol. 1, No. 3, July 1984, pp. 7-45.

Eliminating steps

Barstow, D.R., "Artificial Intelligence and Software Engineering," *Proc. ICSE 9*, IEEE, March 1987.

"Special Issue on Automatic Programming," *IEEE Trans. Software Engineering*, Nov. 1985.

Eliminating rework

Alford, M.W., "SREM at the Age of Eight: The Distributed Computing Design System," *Computer*, Vol. 18, No. 4, Apr. 1985.

Boar, B.H., *Application Prototyping*, John Wiley, New York, 1984.

Boehm, B.W., "A Spiral Model of Software Development and Enhancement," *Proc. IEEE Second Software Process Workshop*, ACM Software Engineering Notes, Aug. 1986.

Boehm, B.W., "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Vol. 1, No. 1, Jan. 1984, pp. 75-88.

Booch, G., *Software Engineering with Ada*, Benjamin/Cummings, 1983.

Mills, H.D., "Structured Programming: Retrospect and Prospect," *IEEE Software*, Nov. 1986, pp. 58-66.

Parnas, D.L., "Designing Software for Ease of Extension and Contraction," *IEEE Trans. Software Engineering*, Mar. 1979, pp. 128-137.

Swinson, G.E., "Workstation-Based Rapid Simulation Aids for Distributed Processing Networks," *Proc. IEEE Simulation Conference*, 1984.

Yourdon, E., *Managing the Structured Techniques*, 3rd ed., Yourdon Press, 1987.

Zave, P., "The Operational Versus the Conventional Approach to Software Development," *Comm. ACM*, Feb. 1984, pp. 104-118.

Zelkowitz, M., and S. Squires, eds., *Proc. ACM Rapid Prototyping Symp.*, ACM, Oct. 1982.

Building simpler products

Agresti, W., *New Paradigms for Software Development*, IEEE Cat. No. EH 0245-1, 1986.

Dowson, M., and J.C. Wileden, eds., *Proc. Second Software Process Workshop*, ACM Software Engineering Notes, Aug. 1986.

Heckel, P., *The Elements of Friendly Software Design*, Warner Books, 1984.

Lehman, M.M., V. Stenning, and C. Potts, eds., *Proc. Software Process Workshop*, IEEE, Feb. 1984.

McCracken, D.D., and M.A. Jackson, "Life Cycle Concept Considered Harmful," *Software Engineering Notes*, ACM, Apr. 1982, pp. 29-32.

Reusing components

B.W. Boehm, *Software Engineering Economics* (Chapter 33), Prentice-Hall, Englewood Cliffs, N.J., 1981.

Jones, T.C., *Programming Productivity*, McGraw-Hill, New York, 1986.

Freeman, P., *Software Reusability*, IEEE Cat. No. EH0256-8, 1987.

Horowitz, E., A. Kemper, and B. Narasimhan, "A Survey of Application Generators," *IEEE Software*, Jan. 1985, pp. 40-54.

Lanergan, R.G., and C.A. Grasso, "Software Engineering with Reusable Design and Code," *IEEE Trans. Software Engineering*, Sept. 1984, pp. 498-501.

Matsumoto, Y., "Management of Industrial Software Production," *Computer*, Vol. 17, No. 2, Feb. 1984, pp. 59-70.

"Special Issue on Reusability in Programming," *IEEE Trans. Software Engineering*, Sept. 1984.

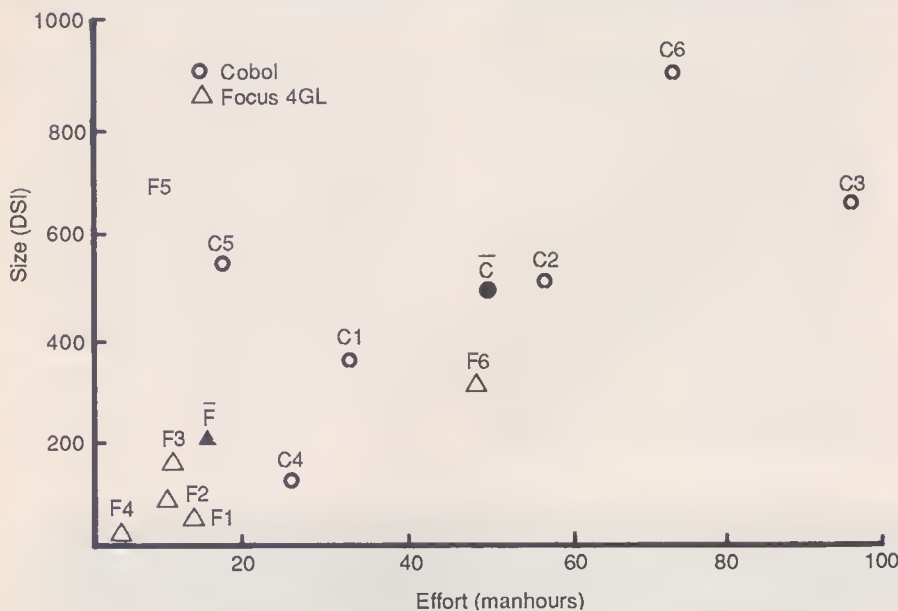


Figure 8. Fourth-generation-language experiment: size and effort comparisons.

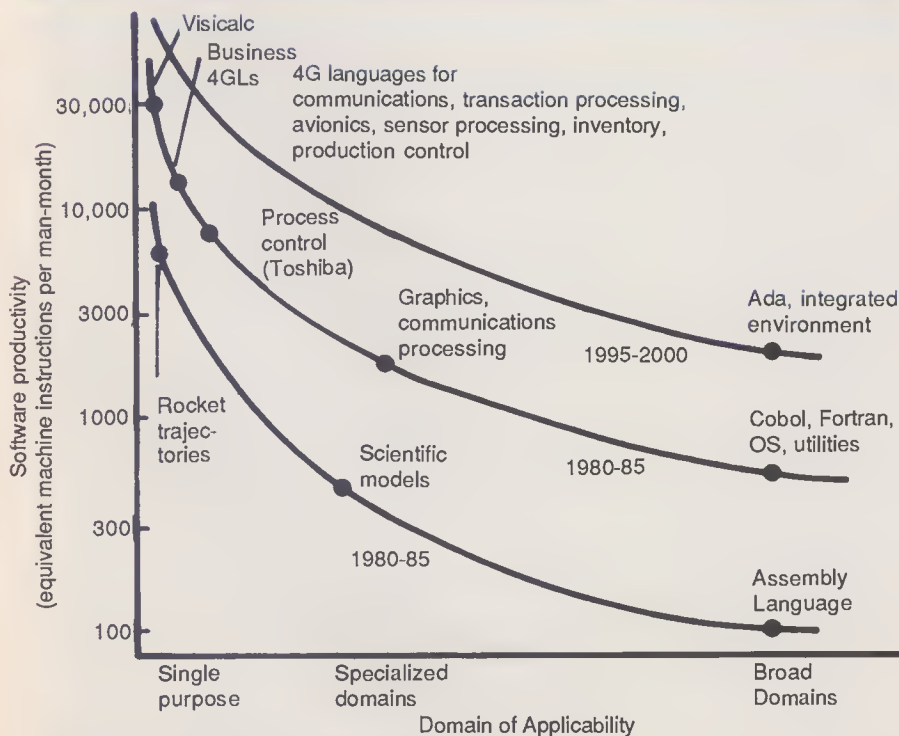


Figure 9. Software technology and productivity trends.

extremely attractive option for significantly improving software productivity, and attempts are underway to create 4GL capabilities for other application areas. Short of a 4GL capability, the other more limited approaches to reusability such as component libraries and application generators can generate near-term cost savings and serve as a foundation for more ambitious 4GL capabilities in the long run.

Software productivity trends

It is difficult to summarize such a welter of issues as those involved in improving software productivity. Figure 9 provides at least a partial summary of some of the key leverage areas. It shows our typical overall progress in improving software productivity between the 1960-65 era and the 1980-85 era, in terms of equivalent machine instructions per man-month, and it projects our potential progress by the 1995-2000 era.

The horizontal dimension in Figure 9 is a qualitative scale indicating the breadth of the domain of applicability of a given software productivity capability. It reflects the fact that our most impressive software productivity achievements to date have been made by exploiting our knowledge of particular application domains.

Thus, for example, even in the early 1960's, when most large, general-purpose systems were being developed in assembly language at a typical rate of 100 delivered machine instructions per man-month (DMI/MM), there were application generators providing productivity rates of 3000 DMI/MM and higher. Several examples were available in the area of rocket trajectory computation, where systems such as Rocket (see Boehm¹⁶) provided a library of reusable components (for aerodynamics, propulsion, guidance and control, earth models, etc.) and specialized extensions of Fortran for users to specify how to link components together to simulate their desired multistage rocket vehicle and flight program.

By the early 1980's, we had progressed in the power and range of domain-specific application generators, so that even higher productivity figures were being achieved in such a variety of domains as spreadsheet calculations (30,000 DMI/MM and up), industrial process control, and business fourth-generation languages. At the same time, we had extended our general-purpose capabilities from assembly lan-

guage and primitive batch operating systems to HOLs with collections of tools providing on the order of 600 DMI/MM for large, broad-domain applications. According to Brooks,¹⁷ those capabilities address the elimination of the "accidental" difficulties in developing software. The domain-specific capabilities shown on the left side of Figure 9 are aimed at reducing the "essential" portion of software acquisition costs.

Thus, for the 1995-2000 time frame, we can see that two major classes of opportunities for improving software productivity exist: providing better support systems for broad-domain applications, involving fully integrated methods, environments, and modern programming languages such as Ada; and extending the number and size of the domains for which we can use domain-specific fourth-generation languages and application generators. Examples of promising future application domains include communications processing, transaction processing, sensor data processing, broader process control areas such as avionics and job shop production control, and broader DBMS-oriented areas such as inventory control and production management.

We have seen that the magnitude and continuing growth of software costs create a strong need to improve software productivity. This implies a need to carefully define software productivity, and since our current productivity metrics are not fully satisfactory, to work on better ones. It also implies a need to develop capabilities that improve not only software productivity but also software quality.

The analyses of software productivity ranges and the software value chain led to the definition of a software productivity opportunity tree which identifies the major opportunity areas for improving productivity:

- Getting the best from people via better management, staffing, incentives, and work environments.
- Developing and using integrated project support environments, which automate portions of the development and evolution process and make them more efficient.
- Eliminating rework via better front-end aids, risk management, prototyping, incremental development, and modern programming practices, particularly information hiding.

- Writing less code by reusing software components, developing and using application generators and fourth-generation languages, and avoiding software gold plating.

As a final conclusion, one point deserves particular emphasis. In pursuing improvements in software productivity, we need to be careful not to confuse means with ends. Improved software productivity is not an end in itself; it is a means of helping people better expand their capabilities to deal with information and to make decisions. Often, helping people to do this will involve us in activities (for example, spending two weeks helping someone find an effective nonsoftware solution to a problem) that don't add points to our software productivity scoreboard. At such times we need to recall that the software productivity scoreboard is just one of the many ways we have to gauge our progress toward better use of computers to serve people. □

Acknowledgments

I would like to thank Frank Belz, Phil Papacchio, Lolo Penedo, and George Spadaro of TRW, and Bill Riddle of the Software Productivity Consortium, for substantial assistance in refining earlier versions of this article, and Bruce Shriver and the *Computer* reviewers for a particularly thorough and helpful job.

References

1. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
2. T.C. Jones, *Programming Productivity*, McGraw-Hill, New York, 1986.
3. B.W. Boehm et al., "A Software Development Environment for Improving Productivity," *Computer*, Vol. 17, No. 6, June 1984, pp. 30-44.
4. M.E. Porter, *Competitive Advantage*, Free Press, New York, 1985.
5. C.E. Walston and C.P. Felix, "A Method of Programming Measurement and Estimation," *IBM Systems J.*, Vol. 16, No. 1, 1977, pp. 54-73.
6. C.C. Green et al., "Report on a Knowledge-Based Software Assistant," USAF/RADC Report RADC-TR-195, Aug. 1983.
7. D.L. Parnas, P.C. Clements, and D.M. Weiss, "The Modular Structure of Complex Systems," *IEEE Trans. Software Engineering*, Mar. 1985, pp. 259-266.
8. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Proc. IEEE Second Software Process Workshop, ACM Software Engineering Notes*, Aug. 1986.

9. B.W. Boehm, T.E. Gray, and T. Seewaldt, "Prototyping vs. Specifying: A Multi-Project Experiment," *IEEE Trans. Software Engineering*, May 1984, pp. 133-145.
10. P. Heckel, *The Elements of Friendly Software Design*, Warner Books, 1984.
11. D.D. McCracken and M.A. Jackson, "Life Cycle Concept Considered Harmful," *Software Engineering Notes, ACM*, Apr. 1982, pp. 29-32.
12. R. Balzer, T.E. Cheatham, and C. Green, "Software Technology in the 1990's: Using the New Paradigm," *Computer*, Vol. 16, No. 11, Nov. 1983, pp. 39-45.
13. E. Harel and E.R. McLean, "The Effects of Using a Nonprocedural Language on Programmer Productivity," UCLA Graduate School of Management, Information Systems Working Paper #3-83, Nov. 1982.
14. T. Guimaraes, "A Study of Application Program Development Techniques," *Comm. ACM*, May 1985, pp. 494-499.
15. C. Babcock, "New Jersey Motorists in Software Jam," *Computerworld*, Sept. 30, 1985, pp. 1, 6.
16. B.W. Boehm, *ROCKET: Rand's Omnibus Calculator of the Kinematics of Earth Trajectories*, Prentice-Hall, Englewood Cliffs, N.J., 1964.
17. F.P. Brooks, "No Silver Bullet—Essence and Accidents of Software Engineering," *Proc. IFIP Congress 1986*, North Holland, 1986, pp. 1069-1076. Also in *Computer*, April 1987, pp 10-19.

A more extensive version of this article, including additional text, figures, and references, is available from the author. To obtain a copy, circle number 182 on the Reader Service Card at the back of the magazine.



Barry W. Boehm is the chief scientist in the Office of Technology at the TRW Defense Systems Group. He is responsible for the group's Ada Office, Technology Education Program, and Quantum Leap Software Development Program. Since joining TRW in 1973, Boehm has held several senior engineering and management positions. He is also an adjunct professor at UCLA and a member of the Computer Society's Board of Governors. Boehm received his BA in mathematics from Harvard in 1957 and his MA and PhD from UCLA in 1961 and 1964.

Boehm's address is TRW Defense Systems Group, One Space Park, R2/2086, Redondo Beach, CA 90278.

Introducing Design.

The first thinking tool for both sides of your brain.

Now there's a powerful graphics and text handling program to help you visualize complex systems.

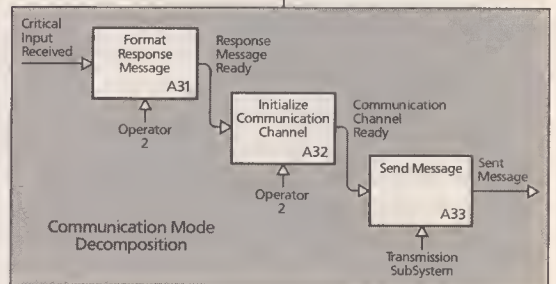
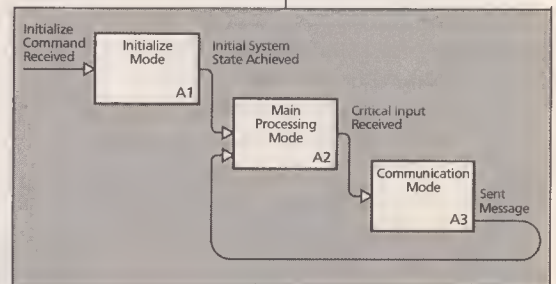
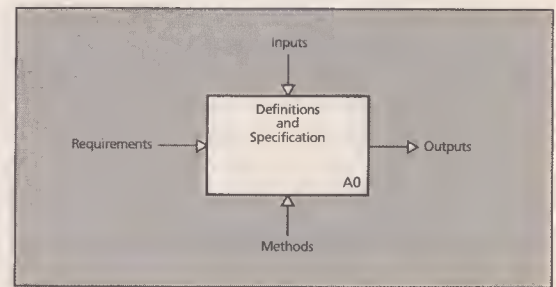
Design lets you create understandable graphic representations of projects and processes on your Macintosh™ or IBM® PC. So you can organize quickly, analyze effectively, and communicate clearly.

Draw flow charts, organizational charts, computer programs, communication networks, presentation graphics and production line processes—in record time. Once you connect one object to another in a diagram, it stays connected, no matter where you move it. *Design* makes it easier to establish, maintain and understand logical relationships.

Build *Design* diagrams up to 999 pages. And arrange them in hierarchical structure. You can also develop successively detailed descriptions within one multi-level diagram. Or hide detail, so the big picture is easier to see. Even edit, manipulate and stylize text inside any graphic object. And create "hypertext" links to organize text across multiple pages.

What's more, you can always upgrade to *Design* development systems from Meta Software. *Design + DA* lets you write Macintosh™ Desk Accessories, mini-applications that extend the power and capabilities of *Design*. And *Design Open Architecture* is a programmable system for developing final applications.

That's the verbal description of what *Design* can do. For a more graphic illustration, look to your right.



Why just draw graphics, when you can draw conclusions?

For more information about *Design*, or for the name of the dealer nearest you, mail this coupon to Meta Software Corporation, 150 CambridgePark Drive, Cambridge, Mass., 02140. If you can't afford to wait, call us at 800-227-4106. In Massachusetts, call 617-576-6920.

Name _____
 Title _____
 Company _____
 Address _____
 City _____ State _____ Zip _____

Meta Software



150 CambridgePark Drive
 Cambridge, MA 02140
 617-576-6920



Design is available for the Apple® Macintosh™ 512, 512E, Plus, SE and II at \$250 as well as the IBM® PC/AT and compatibles at \$350.

IECO 9-87

Apple is a registered trademark and Macintosh is a trademark owned by, or licensed to, Apple Computer Inc. IBM is a registered trademark of International Business Machines Corporation. © 1987 Meta Software Corp.

Reader Service Number 5

A Survey of RISC Processors and Computers of the Mid-1980s

Charles E. Gimarco and Veljko M. Milutinović

Purdue University

datapath

Several years have elapsed since the early reduced computer architecture research was conducted at IBM, Stanford, and Berkeley. In this article, we briefly survey modern reduced instruction set computer architectures. A survey of this type helps illustrate the application areas into which RISC philosophy has moved, the wide range of technologies being used for processor implementation, features that seem to be common with all designs, and unique features introduced to solve specific problems. Finally, although this survey does not provide much depth on each architecture, we will provide interested readers with a more extensive list of references.

Definition of RISC

It seems to be difficult to provide a precise definition of a RISC architecture. The original reduced instruction set computer research, in which an attempt was made to reduce the semantic gap, produced a design philosophy that can be stated as follows^{1,2}:

(1) Analyze target applications to determine which operations are used most frequently.

We briefly survey modern RISC architectures, illustrating common features, range of applications, implementation technologies, and some unique characteristics of today's RISC machines.

(2) Optimize the datapath design to execute these instructions as quickly as possible.

(3) Include other instructions only if they fit into the previously developed

datapath, are relatively frequent, and their inclusion will not slow the execution of the more frequent instructions.

(4) Apply a similar strategy to the design of other processor resources. Include a resource only if it is justified by its frequency of use, and its inclusion will not slow other, more frequently used, resources.

(5) Push as much complexity as reasonable from runtime hardware into the compile-time software.

Within RISC design philosophy, one must be free to make tradeoffs across boundaries of architecture/implementation, hardware/software, and compile-time/runtime.

When a processor design is based upon RISC philosophy, the resulting architecture typically has features in common with other RISC designs. Inclusion or exclusion of particular features should not be used as a measure of a RISC design, however. As you will see in this survey, several computers that are definitely RISC designs have features generally not considered part of a RISC design. Some features commonly seen in reduced instruction set computers are^{3,4,5}

- single cycle execution of most instructions,
- load/store instruction set,

- hardwired instruction decoding,
- relatively few instructions and address modes,
- fixed instruction format for simple decoding,
- complexity pushed into optimizing compiler,
- highly pipelined datapath for much concurrency,
- large register set (windowed or not windowed),
- many levels of memory hierarchy, and
- ✓ instruction set designed for a specific application class.

A brief study of rating RISC designs based upon a similar set of characteristics was done by Tabak⁶ in 1986. This type of rating is useful, but must be carefully applied and interpreted. The danger in using a list such as this is that computers with features contrary to those listed may not be interpreted as RISC designs. For example, some of the surveyed computers have microcoded instruction decoders, large instruction sets, and small register sets, but are still definitely RISC designs. The important issue is that RISC philosophy is followed in the design of a processor for a specific application.

Controversy

Currently there is much controversy over several issues relating to comparison between reduced instruction set computers and complex instruction set computers. The controversy can be divided into two broad categories. First, what differentiates a RISC from a CISC? And second, how does one make reasonable and useful performance measurements for comparisons?

Many of the features now seen in reduced instruction set computers have been extensively used in complex instruction set computers. Some of the features, such as pipelined datapath, caching, and register windowing, are often viewed as attributes of a RISC design.⁷ One advantage that RISC designs have is the existence of a coherent philosophy statement. Originally, RISC designs were targeted for a specific application and therefore optimized for execution of a well defined class of programs. Characteristically, CISCs are designed for a broad range of applications and consequently include support for many different programming environments. It is evident that there is now a trend away from the narrow RISC

application and to a more general purpose design, again confusing the difference between RISC and CISC. RISC proponents often see instruction set size and complexity as a major feature of "RISCy" designs.

Many of the popular performance measurement techniques are of questionable value when comparing RISC performance with CISC performance. Typically, the effects of operating system overhead, compiler optimization, and multiple register sets are not properly considered. Benchmarks related to number of application transactions per second are more meaningful than simply measuring instructions executed per second.

What processors are out there?

Twenty-one processors are included in this survey. Table I presents a list of the surveyed processors with a brief description of each. It is apparent that RISC design has moved out of the university and into the marketplace. Fifteen processors are either currently available for purchase, or will be available this year. Three computers are part of continuing research at Stanford University and the University of California, Berkeley. Three computers are in the design stage and will become embedded signal processors for defense computers.

RISC designs are available as single-chip microprocessors, very large scale integration chip sets with more powerful functions, single-board computers, and superminicomputers. The ARM processor is marketed as VLSI Technology's VL86C010 microprocessor, indicating that some of these processors can appear under different names. A wide range of circuit technologies are used in their implementation. The Whetstone and AMD 29300 processors are bipolar VLSI devices with emitter coupled logic (ECL) internal cells and transistor-transistor logic (TTL) interfaces. Most designs are N-type metal oxide semiconductor (NMOS) or complementary metal oxide semiconductor (CMOS) implementations with geometries of one to three microns. The highest performance processors are being designed for gallium arsenide (GaAs) implementa-

tion. These processors have the longest time to their introduction as a product.

One of the typical characteristics of a RISC design is a certain narrowness of target application. Table I indicates a wide range of individual applications, plus some designs with a very broad application space. Many designs target scientific, engineering, computer-aided engineering (CAE), and computer-aided design (CAD) applications. Almost universally, these use some variant of a Unix operating system. Five of the designs—ARM, Dragon, FAIM-1, SOAR, and SPUR—target symbolic processing and artificial intelligence applications. Most of these are used in multiprocessor configurations. All three GaAs designs and the CAP target signal, data, and image processing applications. The GaAs processors have the advantage of raw speed. The CAP processor has the potential of massive parallelism. The Transputer and MIPS-X processors were designed with general purpose multiprocessing in mind. Four designs—AMD 29300, Spectrum, Ridge 32, and Whetstone—have a broad application target.

Characteristics of GaAs material require changes in architecture when compared to designs in silicon. Of these characteristics, the major design implications are higher off-chip/on-chip delay ratio and lower transistor count per chip. Each of the three GaAs processors address these issues in similar ways, with deep pipelines, processing functions spread over several chips, small instruction sets, and complex memory hierarchies.

This survey is not exhaustive for several reasons. Relatively low cost, high performance computers are an extremely competitive business arena, and many manufacturers will not release details of their current or future products. Several new products are scheduled for introduction in 1987 and will appear between the time of writing this article and its publication. Several manufacturers market products as RISC or RISC-like that we feel do not follow the RISC design philosophy closely enough for inclusion. Other processors are too early in their design phase and will have to be discussed another time, after their designs have stabilized. We included some designs that may not be RISC processors because they exhibit some important RISC features. The Defense Advanced Research Project Agency (DARPA) is funding research for the development of GaAs and CMOS RISC processors. These CMOS processors

Table 1. Description of RISC processors.

Processor	Manufacturer	Configuration	Technology	Application
Accel	Celerity Computing	Single-board CPU and workstation	Uses NCR 32000 MOS chip set	Multiuser scientific and general purpose with Unix support
ARM	Acorn Computers	Microprocessor and single-board computer	2 to 3 μ MOS, 25K transistors	Workstation for HLL programming and real time AI
AMD29300	Advanced Micro Devices	Chip set to construct custom computer	Bipolar LS1 with ECL internal and TTL interface	Construct target architecture without custom VLSI
CAP	IT&T Advanced Technology Center	Slave multiprocessor array	1.2 to 3 μ CMOS, 120K to 600K transistors	Signal processing, image processing, and scientific calculations
Clipper	Fairchild	Single-board computer	3 chips of 2 μ CMOS, total of 836K transistors	Scientific programming in a Unix environment
CRISP	AT&T	Microprocessor	1.75 μ CMOS, 172K transistors	General purpose
Dragon	Xerox PARC	Multiprocessor workstation	2 μ CMOS	Symbolic processing, AI
FA1M-1	Schlumberger, Palo Alto	Multiprocessor workstation	Custom VLSI CMOS	Symbolic processing, AI
MIPS	MIPS Computer Systems	Microprocessor, chip set, and minicomputer	2 μ CMOS, 100K transistors	General purpose programming with Unix support
Pyramid 90X	Pyramid Technology	Superminicomputer	Schottky TTL	Scientific programming and graphics with Unix support
Ridge 32	Ridge Computers	Workstation and superminicomputer	STTL and MOS VLSI in multiple chips	Scientific programming and graphics with Unix support
ROMP	IBM	Microprocessor chip set	2 chips of 1.8 μ NMOS with a total of 111K transistors	Scientific and graphics workstation with Unix support
Spectrum (HP Precision Architecture)	Hewlett-Packard	CPU family	1.5 μ NMOS, 115K transistors	Scientific, business, and instrumentation computing with Unix support
Transputer	INMOS	Microprocessor family	2 μ NMOS, 250K transistors	Multiprocessor
Whetstone 1, Whetstone 11	Integrated Digital Products	Single-board computer, one-chip CPU	Bipolar VLSI with ECL internal and TTL interface	Plug-in replacement general purpose computer
MIPS-X	Stanford University	Microprocessor chip set	2 μ CMOS	Updated MIPS, multiprocessing
SOAR	University of California, Berkeley	Microprocessor and processor board	NMOS, 35K transistors	Smalltalk-80 programming system workstation
SPUR	University of California, Berkeley	Multiprocessor workstation	3 chips in 2 μ CMOS	Parallel processing research in Lisp environment
CDC GaAs	Control Data	Microprocessor chip set	3 chips of 10K gates each in H11L GaAs	Embedded computers for signal processing
McD GaAs	McDonnell Douglas	Microprocessor chip set	E-JFET GaAs, 41K transistors in 2 chips	Embedded computers for signal processing
RCA GaAs	RCA and Purdue University	Microprocessor chip set	ED-MESFET GaAs	Embedded computers for signal processing

Table 2. RISC processors instruction set features.

Processor	Number of Instructions	Decoder	Instruction Length
Accel	142 integer, 126 floating point	Microcoded	90% are 16 bits, 10% are 32 or 80 bits
ARM	?	Programmable logic array	All 32 bits
AMD29000	Variable	Hardwired or microcoded	32 bits
CAP	33 general, 7 scientific, 9 SIMD support	Hardwired; vector units are either 16 or 32 bits	16 or 32 bits
Clipper	101 general, 61 macroinstructions	Hardwired with 2048-word macroinstruction ROM	Multiples of 16 bits
CRISP	33	Decoder unit	16, 64, and 96 bits
Dragon	approx. 150	Programmable logic array	8, 16, 24, or 40 bits
FA1M-1	64	Finite state machine	?
MIPS	?	Hardwired	All 32 bits
Pyramid	90	Microcoded	32, 64, or 96 bits
Ridge 32	170	Microcoded	16, 32, and 48 bits
ROMP	112	Hardwired with 256 words of microcode	16 or 32 packed to 32 bits
Spectrum	140	PLA with millicode	All 32 bits
Transputer	16	Programmable logic array	All 8 bits
Whetstone	18 basic, 181 total	Basic instructions have hardware decode	16 bits
MIPS-X	< 32	Hardwired, distributed	All 32 bits
SOAR	20	Hardwired	8, 16, and 24 bits
SPUR	28 general, 10 Lisp support, 25 floating point	Hardwired CPU and FPU	All 32 bits
CDC GaAs	29 CPU, 31 FCOP, 6 MMU	Hardwired	Multiple; depends on execution unit
McD GaAs	< 64	Hardwired	All 32 bits
RCA GaAs	< 64	Hardwired	32 and 64 bits

are not treated in this survey because they are still early in their development cycle.

Organization of this survey

Since this is a survey of modern RISC designs, we will not discuss the IBM 801, RISC I, RISC II, and SU-MIPS. Also, we included the early commercial products from Pyramid and Ridge only in the tables as a reference. All of these computers have been extensively treated in the literature. We will refer to the early research RISCs when such reference illustrates a common trait, change in design philosophy, or enhancement of a familiar design feature.

This article is organized into sections comparing groups of common features. We investigate the CPU design first, emphasizing instruction set, datapath, and memory system design. Next we investigate the RISC as a system, emphasizing multiple execution units, coprocessor support, multiprocessing, operating system support, language support, and family requirements. Finally, we briefly compare performance.

CPU issues

Instruction set. A tabulation of features related to the instruction set appears in Table 2. The size of instruction sets in this

survey ranges from a minimum of 16 instructions for the Transputer to approximately 268 for the Accel with floating-point support. Even though the instruction set size and instruction length differ for all the surveyed RISC machines, they all use instruction formats that allow rapid decode through use of a consistent opcode field.

Accel designers chose not to limit the processor to too few commands. The instruction set includes 142 integer processor unit instructions and 126 floating-point unit instructions. Ninety percent of the instructions are 16 bits long, the rest are 32 or 80 bits long. This large instruction set is reduced in terms of format and register orientation.

The AMD 29332 ALU has a symmetrical and regular instruction set of byte-aligned and variable-length data manipulation instructions. A complete central processing unit instruction set is constructed out of the arithmetic logic unit instructions, with fields to manipulate registers, coprocessors, and other parts of the CPU. Instructions are limited to 32 bits since all datapaths support a 32-bit word.

The CAP instruction set can be divided into three subsets. Thirty-three instructions execute primarily in the scalar processor. Seven floating-point or scientific instructions are present, and nine instructions are devoted to control of the single instruction, multiple data (SIMD) processor array. Two formats are used, with the 32-bit format split between two vector processors. Vector processors are 16-bit machines.

Fairchild chose to split the Clipper instruction set into two parts, with 101 "simple" instructions forming the basic instruction set. These are all multiples of 16 bits in length. Complex or application-specific instructions may be added to this set through coding into the macroinstruction unit.

The CRISP 33-member instruction set contains instructions of 16, 64, and 96 bits in length. The first bit of each instruction determines length. A major departure from other RISC processors, this design uses memory-to-memory instructions that can execute in one cycle. Also, this instruction set provides direct support of multiplication and division.

Dragon allows variable-length byte-encoded instructions. Consequently, the decode logic is quite complex.

Most of the 64 instructions of the FAIM-I processor can execute in a single cycle.

Like the earlier RISC designs, and in contrast to most modern RISC designs, MIPS has fixed-length instructions, a single address mode, and all instructions 32 bits long. It uses three instruction formats. Like the MIPS processor, Stanford's MIPS-X implements its full instruction set in only 32-bit words. A five-bit opcode field allows only 32 instructions. Multiplies and divides are performed iteratively. It uses only one address mode and four instruction formats.

Most ROMP instructions (79) are 16 bits wide, and the remaining 39 fit into a 32-bit word. Some instructions have both two-byte and four-byte versions depending upon address modes. All instructions are packed into 32 bits upon fetch by the

ROMP CPU.

In the Transputer, single-byte instructions decouple instruction format from the word length of the machine. There are several 16- and 32-bit Transputer models. All instructions are byte aligned with a four-bit opcode field.

To allow compatibility with an existing user base, the Whetstone simulates the Data General Nova instruction set. The manufacturer asserts that the processor uses 18 basic instructions, a subset of the Nova instruction set. All 181 Nova instructions are then mapped into this set of 18, presumably by the compiler. Instructions and data are in 16-bit words.

Each SOAR instruction contains a bit that enables or disables tag checking, allowing conventional programming languages to compile to SOAR instructions. Three instruction lengths (one, two, and three bytes) are used in the instruction set.

All SPUR instructions fit into a 32-bit word, with opcode and register fields aligned. The 55 instructions can be divided into general operations, Lisp support, and floating-point support groups.

The CDC processor instruction set may be divided into three subsets, one for the central processing unit, one for the floating-point coprocessor, and one for the memory management unit. Length of an instruction depends upon where it is executed.

A six-bit opcode field is used for both the McD and RCA processors' instructions. All instructions in the McD processor are 32 bits long, while the RCA processor allows some 64-bit instructions.

In this survey are two basic instruction decoder designs and several interesting combinations. Recall that the original RISC examples all used hardwired logic decoders for the simplest and fastest possible instruction decode. Most of the example processors (12) employ some form of strictly hardware instruction decode. A hardwired decoder, as in CAP, MIPS, MIPS-X, SOAR, SPUR, CDC, and RCA, implies that an area of the CPU chip is designed with minimized logic to decode the instruction set. The MIPS-X processor uses a distributed design with partial decode at the register in which the fetched instruction is placed, and the remainder of the decode at the functional elements in the datapath. The McDonnell Douglas references did not specify instruc-

tion decoder design, but in view of the time required to decode, a hardwired decoder seems the appropriate choice. The FAIM-I employs a finite state machine controlled by instruction opcodes as the method of decode. Three examples (ARM, Dragon, and Transputer) employ programmable logic arrays in the decode path. Use of a PLA is effectively the same as hardwired decode, but does not use minimized logic design. Also, it is much simpler to reprogram a PLA for instruction set modifications than to redesign a hardwired decoder.

Two processors, Accel and Ridge 32, use microcoded instruction decoders and also have the largest instruction sets. The Accel design is built around an NCR32000 CPU and support devices. The CPU is microcodable, allowing specification of a rich instruction set with RISC attributes. The Ridge processors are microcoded, with most instructions executed by a one-instruction microsequence.

The AMD 29300 may be used to implement a RISC CPU with either hardwired or microcoded instruction decode. It requires only the proper control signals for the ALU, registers, coprocessor, and other units in the proper time sequence.

Four processors—Clipper, ROMP, Spectrum, and Whetstone—use a combination of hardwired and microcoded instruction decode techniques. The Clipper has hardwired decode of 101 instructions. A 2048-word macroinstruction ROM is provided, currently with 61 instructions, to allow support of specific programming environments. More macroinstructions may be added by the user. ROMP uses hardwired decode for instruction prefetch and memory data requests. A 256-word microprogram store is used for execution control. Hewlett-Packard's Spectrum processors use hardwired decode for most instructions and a millicode store in virtual memory for implementation of complex instructions. The Whetstone processors use hardwired decode for the basic 18 instructions, with microcode interpretation of the 181-instruction Nova 1200 instruction set.

The CRISP processor employs a separate Prefetch and Decode Unit to break each instruction into fully decoded 192-bit words. These decoded instructions are then loaded into a Decoded Instruction Cache for access by the Execution Unit. The FAIM-I processor employs a separate Instruction Stream Memory to select, format, and deliver decoded instructions to the Evaluation Processor Unit

Table 3. RISC processors datapath features.

Processor	Pipeline Stages	Register Set	Delayed Branch	Execution Units
Accel	3	48 32-bit register windows with overlap of 16, 512 windows	Yes, with delay of 2	Separate integer and floating point units
ARM	3	32 32-bit, 25 for user	No, compiler rescheduling	1 plus fetch incrementer
AMD29300	2	Variable	Yes, plus reorganizer	2: execute and fetch/decode
CAP	?	16 per processor	Send both options, do according to data mask	Scalar unit plus vector data and address units
Clipper	3	16 user, 16 special, 16 supervisor, all 32-bit	?	2: integer and floating point, all on chip
CRISP	3	32 32-bit, as a stack cache	Branch folding	2: prefetch and decode, and execution units
Dragon	?	?	Static branch prediction	2: instruction fetch, execute
FAIM-1	2	Stack	Yes	2: evaluation and switching processors
MIPS	5 (see Figure 1)	32 general purpose, 1 PC, 2 arithmetic, no windows	Yes, with reordering	2: CPU and system coprocessor. Separate data and address units
Pyramid	3	528 32-bit in 16 windows of 64	Yes	Separate instruction fetch and execution units
Ridge 32	4	16 32-bit with overlap windows	Branch prediction	2: instruction fetch, execute
ROMP	3	16 32-bit, 10 system, 4-port register file	Yes, branch with execute instruction	1
Spectrum	5	32 32-bit general purpose	Yes, with reorganization	1 CPU plus coprocessors
Transputer	?	6 32-bit	?	2: execute, I/O
Whetstone	3	4	?	3: arithmetic, address, I/O
MIPS-X	5	32 32-bit general purpose	Yes, with squash instruction, delay of 2	Separate execution and PC
SOAR	3	72 32-bit dual port with 32 register windows, using overlap of 8	Yes, with delay of 1 cycle. Must do type checking.	1
SPUR	4	32 32-bit registers in each window, using overlap of 6 and 10 global	Yes, with cancel compare instruction	1, but instruction and data fetches concurrently
CDC GaAs	6	16 32-bit general purpose	Yes, with reorganization and delay of 2	1
McD GaAs	4	16 32-bit general purpose, 16 32-bit special	Yes, with reorganization and delay of 1	1
RCA GaAs	5 + 2 waits	16 with variable size windows or background loading	Yes, with ignore instruction and delay of 2	Separate execution and PC units

Datapath. Datapath design is quite complex in all the surveyed processors. All 21 examples have pipelines of depth ranging from two stages in the AMD 29300 to seven stages (five plus two wait stages) in the proposed RCA GaAs processor. Generally, the shorter the cycle time, the

deeper the pipeline. This phenomenon is primarily due to two factors. First, all example processors are designed to attempt to begin a new instruction each clock cycle. The MIPS processor can begin any instruction each clock cycle. Most other processors have some subset of

instructions that require use of the pipeline for more than one cycle. Second, the processors have memory access and wiring delays that make up a large fraction of the clock cycle period. Thus, more cycles must be allocated for memory access instructions. Table 3 presents some important

datapath features for the surveyed processors.

The Pipeline Stages column of Table 3 gives the number of CPU pipeline stages, but not details about how they overlap on successive instructions, nor the function of each stage. (Those details are available in the references.) The MIPS processor pipeline deserves special attention because of its complexity. Figure 1 illustrates the pipe for four instructions. It is difficult to give a number-of-stages figure for this design because there are half- and full-cycle stages, and concurrent activities within each instruction execution. The clock cycle is 60 nanoseconds divided into two 30-nanosecond phases. Complexity is partially due to the requirement to execute all instructions in one cycle.

Several register set designs were used, with most processors using an organization with 16 to 32 32-bit registers. All the surveyed RISC designs attempt to gain performance advantages through (1) the speed of an on-chip variable store and (2) the compiler's ability to effectively use locality of program variables. Both windowed and unwindowed register files occur. Thirteen processors have register files with no windowing. Several nonwindowed designs, and even windowed designs such as SOAR, use *load multiple* or *store multiple* instructions to speed context switches. Table 3 gives window size details of the five windowed examples.

The ARM processor resembles the Pyramid design in the sense that it employs an extremely large number of registers. A few processors, such as the ROMP and AMD 29300, have *four-port register files*, allowing multiple reads and writes in a single clock cycle. The AMD allows expansion of the register file in length and width by adding more register chips. The Transputer has an extremely small register file. Even though only three registers are available for ALU operations, the Transputer maintains a workspace pointer in the register stack pointing to the first variable in memory. A delay of one cycle is thus incurred at each variable access. MIPS-X maintains 33 bits on each variable in the register file. The 33rd bit allows a read and write to be performed in a single cycle. The CRISP and FAIM-1 processors use a stack-based register file organization. In CRISP, a separate stack cache appears to the compiler as memory, is arranged as a circular buffer, and allows two reads and one write per cycle. Processors using stacks do not have the register allocation

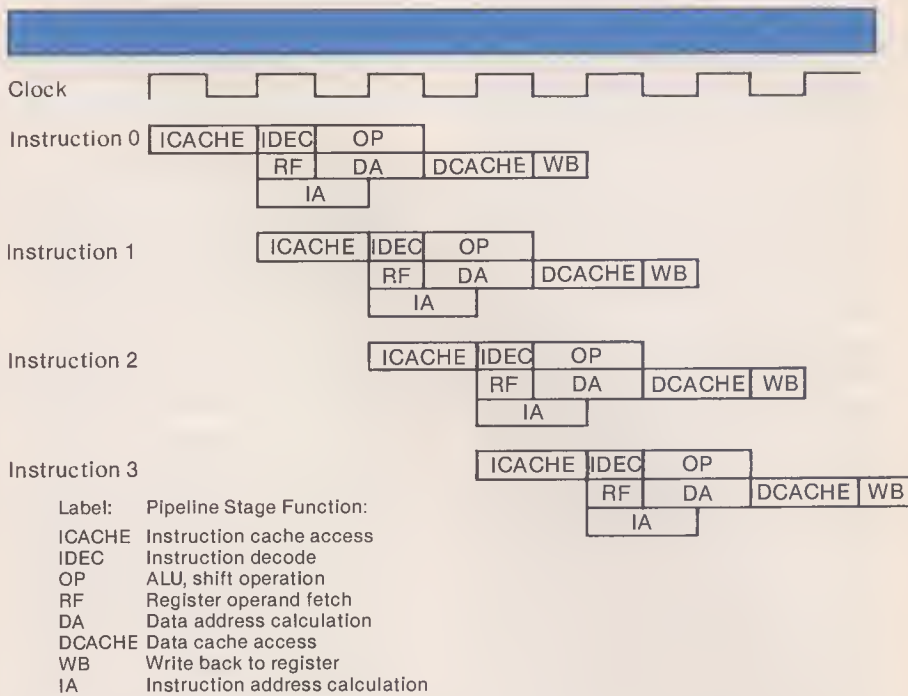


Figure 1. MIPS computer systems pipeline.

problems that other RISC processors' compilers have.

Most of the example processors divide CPU tasks into subtasks for separate execution units. A greater degree of concurrency is allowed at the expense of increased chip resource requirements and more complex control. Twelve processors (ARM, AMD, CAP, CRISP, Dragon, MIPS, Pyramid, Ridge, Transputer, Whetstone, MIPS-X, and RCA) have separate units for concurrent ALU execution and I/O operations. In most processors the I/O unit includes a separate ALU for address computation. The CAP divides execution into a scalar unit and separate vector data and vector address units. The Transputer I/O units are unique in providing four bidirectional links to other Transputers. The I/O unit coordinates the queueing of transmitted and received messages. The Whetstone divides tasks even further with separate address and I/O units. The SPUR processor is unique in performing independent and concurrent data and instruction fetches. Two logically separate units (prefetch and decode, and execution) in the CRISP are connected by a decoded instruction cache. This organization allows each unit to operate autonomously, with no central controller. Many processors use coprocessors to perform complex operations such as caching, memory man-

agement, and floating-point computations.

Several different shifter designs are included in the datapath of RISC processors. SPUR uses a simple unit that allows one-bit right or one-, two-, and three-bit left shifts. Due to timing constraints, RCA requires two instructions to perform a shift. The first instruction loads data into the shifter. A second instruction executes the shift and writes back results. Other processors use barrel or funnel shifters.

As in the original Stanford MIPS, some of the surveyed processors do not have hardware to manage pipeline interlocks. The processors with stated software interlocks are MIPS-X, MIPS, McD, RCA, and Spectrum. Of these, all but Spectrum appeared to be influenced by the SU-MIPS design.

Three processors (ARM, Clipper, and CDC) followed the Berkeley lead by providing hardware interlocks. Most processors provide register bypass paths to allow availability of a result of a computation for the next instruction.

Most processors use delayed branching to accommodate the condition when a branch is not taken. The ARM processor does not allow delayed branching. Three processors (CRISP, Ridge, and Dragon) use branch prediction in the compiler to increase the efficiency of delayed branch-

ing. The compiler sets or clears a bit in the branch instruction depending upon the likelihood of a branch taken. In CRISP, the technique of "branch folding" allows most branches to execute in zero time. This technique makes use of a dynamically generated "next address" field in each decoded instruction. In most processors, the compiler or reorganizer moves code to fill one or two slots following a branch with something more useful than *nop* instructions. The deeper the pipeline, the more difficult it is to fill the slots. For example, the deep pipelines of CDC and RCA processors have two delay slots to fill with code reorganization.

An interesting feature added to the MIPS-X processor consists of two versions of branch instructions. For a nonsquashed branch, instructions in the delay slots always execute. A squashed branch is used when both delay slots are not filled. Branch destination instructions are put into the slots and changed to *nop* instructions if the branch is not taken. SPUR performs a similar operation with its cancel compare instructions. To keep the processor array synchronized in CAP, instructions for *branch-taken* and *branch-not-taken* are transmitted by the scalar processor to the array processors. Computation then continues first with the taken instructions, then with the not-taken instructions, according to a branch condition mask.

Memory system. Because of the need to keep instructions and data supplied to the processors, RISC memory systems are necessarily quite complex. Several levels of memory hierarchy are used, often with data and instructions separated. Virtual memory is universally supported in the surveyed processors.

A hierarchical memory is one in which parts of the total storage space exist at different physical distances from the CPU, and at different operational speeds. It is currently not possible to build all the desired memory at a speed in which it can be accessed by the pipeline without delay. Several memory structures are possible, but the most common includes an on-chip instruction buffer of sufficient size to hold the next few instructions. This buffer is kept full by prefetch logic. Next is an on-chip or off-chip instruction and/or data cache. Some processors cache both data and instructions, others have just one or no cache. Finally, main memory exists off-chip and often off the processor board. Of course, access time increases as distance

from the CPU or size of the memory system increases.

Six processors use on-chip instruction prefetch buffers in addition to instruction or data caches. CAP uses RAM in the vector processors as local instruction storage. Clipper, Transputer, and Dragon use a four-word instruction buffer. SPUR has a 512-byte prefetch buffer (used as cache) with a reported 98 percent hit ratio. On-chip instruction or data caches are implemented in five of the surveyed processors. ARM has a shared instruction and data cache. Future plans include splitting the on-chip cache for greater flexibility. CAP also has 256 kilobits of local RAM used to cache instructions and data in each vector processor. Each Dragon processor has an on-chip fully associative cache. The Transputer has one kiloword of on-chip memory to store instructions and data. MIPS-X processor has a 512-word on-chip instruction cache.

All the processors without on-chip cache use some form of off-chip cache for data or instructions. The Accel processor board uses three distinct on-board cache systems. A 64-kilobyte cache holds instructions. To support virtual addressing, a 32-kilobyte four-way set associative address translation cache is used. And a 64-kilobyte stack register cache is used for expansion of the CPU registers.

Separate MMU chips in the Clipper implement separate external data and instruction caches. Each cache contains four kilobytes of two-way set associative memory. The instruction cache is reported to have a 96 percent hit ratio. A program counter inside the instruction cache MMU reduces CPU-to-MMU bus bandwidth requirements. The data cache supports copy back, write through, and noncacheable control schemes. With two MMU chips, instruction and data access can occur concurrently.

The MIPS processor uses separate instruction and data caches of up to 16 kilowords each. Cache control for both, and memory management, are on the CPU chip, while memory resides off-chip. A write buffer chip in MIPS allows up to four writes to be queued, freeing the CPU.

Ridge processors use a 256-byte instruction cache and no data cache. Spectrum uses separate instruction and memory caches, both located on the CPU board. A

three-level cache scheme is implemented in the Whetstone processor. The CPU directly accesses 128 kilobytes of very fast main memory. Separate on-board and off-board cache memories allow physical memory expansion up to 32 megabytes. Cache memory is updated through a separate DMA interface with the system disk.

In addition to an instruction buffer, MIPS-X uses an expandable 16- to 64-kiloword external direct mapped instruction cache. This cache is designed to minimize memory bus traffic.

Separate 128-kilobyte instruction and data caches are located on the SPUR processor board. The Berkeley Ownership algorithm insures cache consistency in a multiprocessor environment. All three GaAs processors use separate instruction and data caches. Due to chip resource constraints, the caches must be located off-chip. The CDC processor, like Clipper, uses two MMU chips to separate instruction and data cache control. Each CDC cache is one kiloword, direct mapped.

System issues

Division of functions. All the surveyed processors reflect different approaches to the design of reduced instruction set computer systems. Many of the processors divide system functions across chip boundaries, allowing integration of a large amount of functionality. Almost all these processors provide floating-point support utilizing the IEEE 754 standard. Floating-point support usually is provided by a separate coprocessor chip.

The Accel processor is available through Celerity computers with one or two floating-point units. The floating-point unit is physically separate from the CPU and loads data and instructions through the CPU registers.

As a user configurable system, the AMD processor allows connection of separate ALU, multiplier, and floating-point units, encouraging inclusion of only necessary functions. Each unit is a separate VLSI chip.

CAP is unique in that it implements an array of 20 identical RISC processors with memory on a single integrated circuit. This array chip employs software-based fault tolerance, necessary with such an extremely large device, by disabling faulted processors or memories. A multiprocessor array can be constructed from one or more CAP chips, depending upon the application. A scalar execution unit

fetches instructions and provides control for an array of CAP chips that comprise the parallel execution unit. Each processor on the array chip has a microinstruction set similar to other bit slice processors, containing *and*, *add*, *or*, and *sub* type instructions. CAP may be treated as a collection of SIMD processors, each operating in synchrony on its own data.

The Clipper processor board uses three chips of two different functions. The CPU/FPU chip contains execution circuitry. Floating-point support is included on the same chip as the integer execution unit to avoid delays associated with moving floating-point operands between chips. Two combined cache and memory management unit chips are used on the Clipper board. One CAMMU chip manages instruction cache and memory access, and the other one manages operand cache and memory. This arrangement allows overlapped instruction and data access. Floating-point operations are performed on-chip for greater performance. This eliminates moving instructions and data to a coprocessor.

The Dragon processor is used in a workstation of one to ten identical processors with a shared memory. Cache in each processor mediates between processor and memory, reducing bus traffic—typically the bottleneck in shared memory systems.

FAIM-I is a fully distributed multiprocessor system with no shared memory. Each processing element, called a Hectagon, contains an Evaluation Processor, Switching Processor, Post Office (topology dependent hardware), Instruction Stream Memory, Scratch Memory, and Associative Memory. Hectagons are connected together into hexagonally arranged Surfaces. Surfaces can also be connected into larger structures. The memory system is a highly parallel associative store. FAIM-I is used as a hardware accelerator unit attached to a host Lisp processor. The prototype FAIM-I will have 19 processors, but the system can be scaled to an arbitrarily large number of processors.

The MIPS system is implemented as a three-chip set. A CPU chip contains all integer execution, memory management, and cache control. A tightly coupled floating-point processor supports IEEE single- and double-precision operations. A write buffer chip provides queuing of memory access. This allows the CPU to continue its task without waiting for a write to complete.

The IBM ROMP processor is implemented in two devices, a CPU chip and a

separate memory management unit. CPU and MMU communicate over a 32-bit parallel packet switched channel. A floating-point accelerator board is available for inclusion in the IBM RT PC system.

The Spectrum processor is designed to support a wide range of applications. Three kinds of assist processors are available. Coprocessors may be added for graphics or floating-point support. Special function units can be added to perform fixed-point arithmetic, encryption, emulation, and other functions. Finally, multiprocessing units can be added to the system.

Transputer is a RISC processor for parallel processing. Each processor has four dedicated 10 megabit-per-second interfaces to other Transputers or peripherals. Other processing units, such as communications link adapter, graphic, or floating-point, can be added to the system. Multiprocessor performance can scale linearly with the number of processors.

Whetstone uses three independent processors to perform arithmetic/logic, memory/address, and DMA-I/O operations. An optional floating-point unit is also available.

MIPS-X differs from the original SU-MIPS in the inclusion of coprocessor support. MIPS-X is designed as a processor in a shared memory multiprocessor system. To that end, memory bus traffic is minimized by keeping the cache miss ratio very small through use of a large (16- to 64-kiloword) external cache. Simulations indicate that six to eight processors may share the same memory without noticeable performance degradation due to bus contention.

The SOAR processor is implemented as a board for use with a Sun workstation. The SPUR processor follows development of RISC I, RISC II, and SOAR at Berkeley. It provides support for Common Lisp and IEEE 754 floating-point standards in a parallel processing environment. SPUR is a general purpose processor with some Lisp support. Three types of chips comprise the majority of the SPUR processor. A CPU performs integer execution and instruction fetch. A cache controller chip manages the memory and the instruction cache. A separate FPU implements the IEEE standard. The FPU tracks CPU instructions and fetches

required instructions from the instruction cache.

The CDC system uses four major integrated circuits. A CPU executes most instructions with a RISC processor. Up to four different coprocessors can be used to augment the instruction set, with execution capabilities for specific applications. A floating-point coprocessor (FCOP) is being developed. The operand and instruction caches are each controlled by their own memory management unit. System devices interconnect through 32-bit parallel operand and instruction data buses, a 24-bit parallel instruction address bus, and a 26-bit parallel operand bus. Each of the system chips is pipelined.

The McD GaAs processor design was derived from the SU-MIPS microprocessor. Modifications include reduction in transistor count and complexity, and the addition of two coprocessor interfaces for floating-point operations. The CPU performs all fetch operations. The FPU loads all instructions in parallel with the CPU, executing only the ones that apply to floating-point operations. A system controller chip contains interrupt management, clock generation, and low-speed I/O operations.

Operating system/language support.

The dominant operating system supported on new RISC processors is Unix or one of its variants. The ARM, Clipper, MIPS, Pyramid, Ridge, and Spectrum processors all have applications that run under a resident Unix-type operating system. The Unix processors all support some subset of the C, Pascal, Fortran, Ada, and Cobol programming languages. Optimizing compilers exist for all these languages.

The ROMP processor was designed to support IBM's PL.8 compiler. The two Berkeley processors, SOAR and SPUR, support Smalltalk and Lisp. SPUR was designed to also allow execution of untagged languages. Dragon was designed primarily to execute Lisp, but also provides support for Cedar and Smalltalk. The FAIM-I processor was designed around the OIL intermediate language, a high-level symbolic processing language. CRISP is based upon the Bell Labs C Machine, and was optimized for execution of programs written in the C language.

All three GaAs processors were designed for high level language execution. The CDC specifically supports, through the development of optimizing compilers, the Pascal and Ada languages. Whetstone represents a departure from the norm of

Table 4. RISC processors performance.

Processor	Cycle Time/Clock Rate	Instruction Rate (MIPS)
Accel	100 ns	3.2 MIPS
ARM	8 MHz	3-4 MIPS
AMD29000	125 ns	4-5 MIPS
CAP	I: 10 MHz, II: 25 MHz	12.5 MIPS peak, scalar unit
Clipper	33 MHz	5 MIPS
CRISP	16 MHz	> 10 MIPS
Dragon	10 MHz	5 MIPS per CPU
FAIM-I	?	?
MIPS	16.6 MHz	8 MIPS
Pyramid	125 ns	2-4 MIPS
Ridge 32	125 ns	1-4 MIPS
ROMP	170 ns	2 MIPS
Spectrum	30 MHz	10.8 MIPS
Transputer	50 ns	10 MIPS
Whetstone	50 ns	5-13.3 MIPS
MIPS-X	20 MHz	> 10 MIPS
SOAR	400 ns	See text
SPUR	150 ns	?
CDC GaAs	5 ns	91 MIPS
McD GaAs	10 ns	100 MIPS
RCA GaAs	200 MHz	200 MIPS peak

this survey by supporting an existing user base in the RDOS, SLICE, and IRIS systems.

Family requirements. A few processors were designed under the constraint of compatibility with an existing family of processors. The HP Spectrum gained some complexity by the requirement of object code compatibility with existing Hewlett-Packard computers. This level of compatibility carries with it a great marketing advantage, since users can experience at least a two times performance increase with existing applications by changing hardware.

The Whetstone design was completely constrained by the requirement of assembly code compatibility with Data General Nova I200 computers. This provides a reduced architecture product that is software compatible with an existing user base. Both Ridge Computers and Pyramid Technology are working on new additions to their existing RISC-based computer products.

Advanced Micro Devices has announced a new AMD 29000 RISC processor. This 25-megahertz device, with 192 registers configured as a stack cache, a four-stage pipeline, provisions for macroinstructions (similar to the Clipper), and

single-cycle instruction execution, will be the next step beyond their current AMD 29300 series devices.

Performance notes

The processors in this survey can be divided into two performance groups. The largest group contains those processors with silicon implementations. From Table 4 we can see that most processors have cycle times in the range of 30 nanoseconds to 400 nanoseconds. The data in the Instruction Rate column are, where possible, average processing rates from measured or simulated benchmarks, and not peak processing rates. Transputer's 10 MIPS rate may be misleading, since it relies on the assumption that all instructions and operands reside in the on-chip RAM, with no external memory delays. Overall processing rates of multiprocessors such as SPUR, Transputer, and Dragon are scalable on the number of active processors.

A second performance group contains the gallium arsenide processors. These three processors are designed for a 200-megahertz clock. The clock rates and instruction rates are an order of magnitude

greater than those possible from silicon processors. The difference in instruction rate figures for the CDC and McD processors is most likely due to the measurement method.

SOAR's performance is compared to other Smalltalk systems, the Xerox Dorado, and the VAX 11/780. Benchmarks show SOAR operating at 50 percent of the Dorado rate and about six times faster than the VAX. Choice of benchmark has a tremendous effect on performance measurement, since SOAR is a language specific processor.

Reduced instruction set computers have quickly moved into many different application areas, indicating that the RISC philosophy can be applied to special purpose processors as well as to large general purpose computers. Most of these computers were designed for engineering and scientific computing. Few of the surveyed processors possess every characteristic commonly attributed to RISC designs. Most share some CISC characteristics, providing additional processing capacity for a given application. In fact, it is interesting to note that every processor surveyed contained architectural features typically attributed to CISCs, and every CISC feature was represented in at least one RISC design, indicating that in future designs good and useful architectural concepts will survive.

It should be obvious that an optimizing compiler is an integral part of any RISC design. Most RISCs use some form of delayed branching and also require code reorganization for optimum performance. The development of optimizing compilers and reorganizers often lags behind development of new computing hardware. One must be certain these tools exist for the language in which their applications programs are to be implemented.

We can expect to see more introductions of RISC machines in 1987 and beyond. Several of the recently introduced processors have begun to find their way into commercially available systems (such as Clipper in Opus Systems personal mainframe computers, Integraph workstations, and an IBM PC accelerator card). We can expect this trend to continue as software support of these processors matures. □

Acknowledgment

This research has been partially supported by NCR.

References

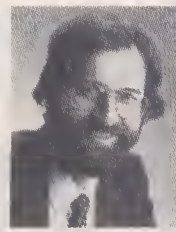
1. D.A. Patterson and J. Hennessy, "Response to 'Computers, Complexity, and Controversy,'" *Computer*, Nov. 1985, pp. 142-143.
2. M.G.H. Katevenis, *Reduced Instruction Set Computer Architecture for VLSI*, PhD thesis, University of California, Berkeley, Oct. 1983.
3. R.P. Colwell et al., "Computers, Complexity, and Controversy," *Computer*, Sept. 1985, pp. 8-19.
4. J.R. Mashey, "RISC, MIPS, and the Motion of Complexity," *Proc. Uniforum*, Feb. 1986.
5. M.E. Hopkins, "A Definition of RISC," *Proc. Int'l Workshop on High-Level Architecture*, 1984, pp. 3.8-3.11.
6. D. Tabak, "Which System is RISC?," *Computer*, Oct. 1986, pp. 85-86.
7. N. Mokhoff, "New RISC Machines Appear as Hybrids With Both RISC and CISC Features," *Computer Design*, April 1, 1986, pp. 22-25.

Additional references for this article are available by using the Reader Service Card. Circle number 183.



Charles E. Gimarc is currently a doctoral student in electrical engineering at Purdue University. He has held design engineering positions at Texas Instruments and Ampex Corp. He has also been a computer systems engineer in the Digital Signal Processing Laboratory at the Georgia Institute of Technology. His research interests include computer architectures for digital signal processing.

Gimarc received a BSEE degree from Texas A&M University and an MSEE from the Georgia Institute of Technology. He is a student member of the IEEE and the Audio Engineering Society.



Veljko M. Milutinović is on the faculty of the School of Electrical Engineering at Purdue University. He has published over 60 technical papers, two books, and four edited books. His current interests include VLSI computer architecture for GaAs, high-level language computer architecture, and artificial intelligence computer architecture. He has consulted for a number of high-tech companies and is currently involved in the industrial implementation of a 32-bit VLSI microprocessor for GaAs technology, with responsibilities in the microarchitecture domain.

Milutinović received his PhD from the University of Belgrade in 1982. He is a senior member of the IEEE and is on the Euromicro board of directors.

Readers may write to Milutinović at the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

Software Engineers

Northrop Corporation's Defense Systems Division in Rolling Meadows, Illinois is the fastest-growing enterprise in an expanding electronic countermeasures industry. We offer professionals with a BSCS, BSEE, BS Math or Physics (or equivalent) MS preferred, and a minimum of 3 years experience, opportunities in the following areas. **Management, System Architect, Technical Leaders and engineering assignments available.**

System Programmers

Our many varied applications require significant growth in our support capabilities. We need the best people with experience in:

- LANGUAGES, including Ada, Assembler, C, FORTRAN, JOVIAL, and Pascal
- OPERATING SYSTEMS, including UNIX and VMS
- Development of Real-Time Operating Systems
- Development of Software Tools
- Performance Modeling and Evaluation
- Use of Software Structured Development Methodologies

Software Systems Engineers

Our software engineers develop software from system requirements through implementation, and need experience in:

- Software Requirements Analysis • Architectural Design
- Software Validation and Test Specification
- Performance Specification and Modeling
- Interface Design and Specification

ECM/EW Systems Software Engineers

ECM/EW Systems are our business. We need the best people with experience in:

- Real-Time Control Systems
- Radar Data Processing
- Object Discrimination & Classification

- Embedded Computer Systems
- System and Unit Level Diagnostics

- ECM Algorithm Development
- Calman Filtering
- Optimal Control

Hardware Diagnostics Software Engineers

We design and develop advanced systems using the latest hardware and software technologies for our military clients. Experience required:

- Intelligent Control Panel Systems
- Development
- Micro and Macro Diagnostics for Fault Identification
- Built-in-Test
- Functional Test

Artificial Intelligence

Artificial intelligence technologies promise state-of-the-art solutions to complex ECM/EW challenges. Positions require people who can bring AI technologies to avionic electronics, with experience in:

- LANGUAGES, including: Ada, C, LISP, and Prolog
- System Prototyping
- Implementation of AI Technology in Real-Time Embedded Systems
- Knowledge Engineering
- Expert Systems Development

Interested individuals are encouraged to forward resume to: **James Frasca, Technical Recruiter, Dept. C91, Northrop Corporation, Defense Systems Division, 600 Hicks Road, Rolling Meadows, IL 60008.** We are an equal opportunity employer M/F/V/H. U.S. Citizenship Required.

NORTHROP

Defense Systems Division
Electronics Systems Group

Define Tomorrow at BDM.

If you value professional challenge and want to grow, join a company that's shaping the future: BDM. We're one of America's foremost technical and professional services firms with a sustained growth rate of 30% per year. And we're committed to strengthening the nation in meaningful ways through our efforts in defense, communications, space, and other vital areas.

Using state-of-the-art technologies, you'll take your capabilities directly to our clients, working to create effective new solutions to difficult problems. You'll work in an environment that's fast-paced and innovative — designed especially with your entrepreneurial spirit in mind.

Exciting technical challenges exist now in the following BDM locations:

McLean, VA
Albuquerque, NM
Los Angeles, CA
Columbia, MD
Dayton, OH

Seattle, WA
Tacoma, WA
Huntsville, AL
Austin, TX

We're looking for skilled professionals, preferably with an MS or PhD degree, with experience in the following areas:

ADVANCED COMPUTING TECHNOLOGY

- Expert Systems Applications
- Natural Language Processing
- Parallel Computing Applications
- Optical Computing

SIGNAL/IMAGE PROCESSING

Algorithm & SW Development for:

- SAR & Acoustic Imaging
- Sensor Modelling
- Restoration and Enhancement
- Classification & Tracking

DATABASE MANAGEMENT SYSTEMS

- ORACLE
- IDMS/R
- M204
- DATACOM DB, ADR Products
- INGRES
- Database Security
- Relational Database Design

MANAGEMENT INFORMATION SYSTEMS

- COBOL, ISPF
- IBM Mainframe
- Large-Scale MIS Development
- Project Leadership



SOFTWARE ENGINEERING

- SW Test & Evaluation
- SW IV&V
- Optical Disk Storage Technology
- Computer Security
- Ada, Pascal, Fortran, Lisp, CMS 2, Jovial
- UNIX, 'C'
- Ethernet, Decnet
- Zenith Micros
- Embedded SW T&E
- Decision Support
- SW/HW Integration
- Software Requirements Definition & Design
- SW Graphics
- Sun Workstation/Internetworks
- Computer Integrated Manufacturing

If you have experience in any of these areas, we would like to talk to you. Please send your resume and geographical preferences in confidence to: BDM International, Inc., Holly K. Marcario, Dept. CM987, 7915 Jones Branch Drive, McLean, VA 22102. An Equal Opportunity Employer. U.S. Citizenship Required.

BDM

And Now a Case for More Complex Instruction Sets

Michael J. Flynn, Chad L. Mitchell, and Johannes M. Mulder
Stanford University

With the spate of recent papers and product announcements,^{1,2} it might seem that the reduced instruction set computer, or RISC, approach to instruction set design has been universally accepted as superior. These RISC designs³ have been characterized as having few simple instruction types with fixed instruction size and formats. The antithesis of RISC designs have been designated CISC, for complex instruction set computer, and characterized as having large instruction vocabularies with multiple sizes, formats, and addressing modes.⁴

RISC performance estimates, when compared to alternative conventional CISC (such as VAX, S/360, etc.) approaches, seem impressive. While there have been critical appraisals of the RISC approach,⁵ the comparative performance evaluations provide formidable, although qualified, evidence in its favor.

Our purpose here is to evaluate and compare RISC-type designs with non-RISC instruction-set extensions using a level playing field: with similar compiler strategies, without compatibility considerations, and with similar implementation constraints. We also deal with instruction set evaluation. The data presented is based on five benchmark programs discussed later.

**Using a computer
architecture
simulation platform,
we can perform
instruction set
tradeoffs with a
common optimizing
compiler and
workload.**

Instruction sets have many attributes as well as constraints. The key to careful instruction set evaluation is to consider key attributes and the tradeoffs possible among them in light of implementation constraints.

Modeling performance in instruction set design

It is difficult to create a truly fair basis for comparing instruction set designs⁶

because of the myriad of considerations and compromises in achieving the final design. Broadly, these factors fall into two classes: those concerned with functional requirements and those directly related to performance.

The former class includes issues such as compatibility, design time, and technology selection. We will discuss it briefly in a later section.

Performance-oriented considerations. Quantitative evaluation certainly makes for interesting comparisons, especially when driven by a common and representative workload. Typical runtime measurements include both static and dynamic characteristics of processor execution:

(1) Static measures. They simply represent the static size of program representation. In the absence of other considerations, smaller size is better, as more concise code should have better locality in a memory hierarchy and requires less memory bandwidth.

(2) Dynamic measures. They include the number and type of instructions executed, number of data references required for reads and writes, and memory traffic as measured in number of bytes transferred.

(3) Compilation time. An item frequently overlooked in comparisons is the time it takes to create a program represen-

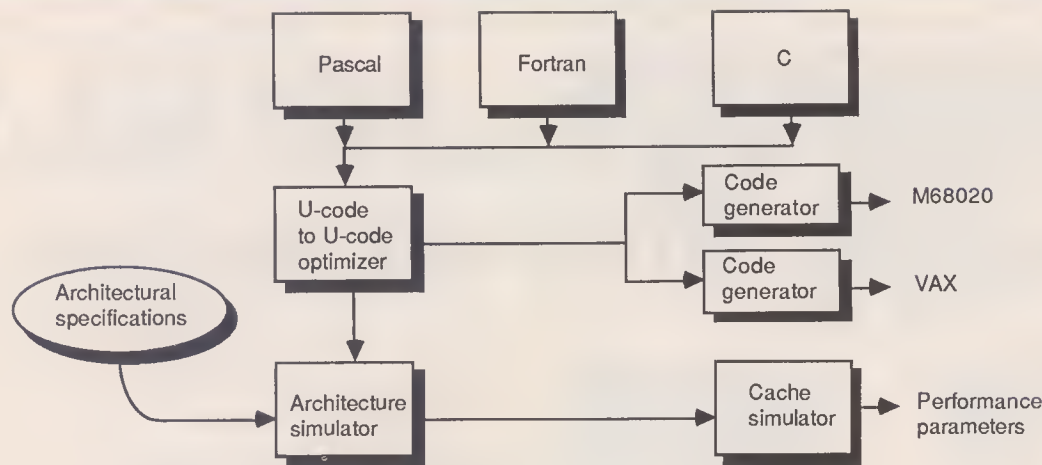


Figure 1. Computer architect's workbench.

tation. It has been variously estimated that half of mainframe problem-state activity involves compiling programs.

(4) Operating system execution time. Depending on the system and the application, a significant fraction of machine execution time is spent in the operating system. A good instruction set must support system functions.

A basis for comparison. In the creation of a new instruction set design, the evaluation of expected runtime parameters is an important facet in understanding the required tradeoffs. While it is imperative to compare a projected design against existing designs, it is just as important to understand the limitations of such comparisons. To achieve a small runtime advantage, for example over a VAX model, is necessary but not conclusive evidence of design validity. Similarly, comparing a design and technology which will be available to customers in future years with designs done several years ago and currently in production may give a misleading impression of the role of the instruction set in determining performance.

A further problem is that comparisons are made not simply across processors, but usually across processor-compiler pairs. Processor-instruction-set variations can become lost in the noise of differences in compilers. We need to create a level play-

ing field for the evaluation of instruction set alternatives using a common compile-time strategy.

In the remainder of this article we assume

- a common workload (benchmarks),
- a simple register-oriented base instruction set,
- a fixed implementation technology,
- similar compiler optimizations for all instruction set variations, and
- the same arithmetic logic unit (ALU), data paths, and instruction vocabulary for operations for all instruction set variants (thus, the same ALU cost). Instruction count differences will arise *only* due to format and register differences.

We will use a simple base design to evaluate the usefulness of several possible additions.

A computer architect's workbench

The computer architect's workbench⁷ is a set of tools developed at Stanford which allows the evaluation of architectural and memory system parameters for a variety of different instruction sets using a common compiler front end. As shown in Figure 1, applications written in Pascal, Fortran, or C compile into an intermedi-

ate code called U-code. If desired, we can optimize the U-code representation of the application by means of the global U-code to U-code global optimizer.

The actual simulation consists of a static and a dynamic part. During the static part, the simulator extracts static information per *basic block* (a code segment with a single entry and exit point). The information includes the code size of the basic block for the target architecture and data-reference information. The architectural specification drives this stage and determines the code size. To examine the relative performance of different instruction sets on an application, an architect simply parameterizes an architecture or instruction set family.

During the dynamic part, the benchmark executes and passes dynamic information to the simulator. Information includes the currently executing basic block and dynamic data references. The simulator associates the dynamic basic-block trace with the static basic-block information; generates simple architectural characteristics such as program size, number of executed instructions, number of memory references, and so forth; and generates an address trace to drive a subsequent cache simulator.

The architect's workbench allows rapid evaluation of multiple architectural and memory system designs. Among the instruction sets currently available are \

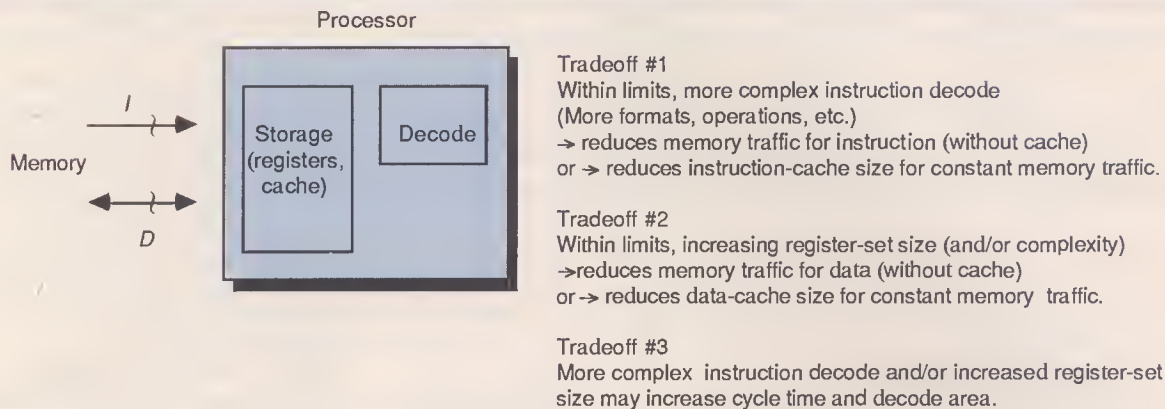


Figure 2. Basic tradeoffs.

- stack machines, including fixed-size stack machines, byte-encoded stack machines, and B6700-type stack machines;
- register-set machines, including load-store architectures and System/360-type architectures; and
- direct-correspondence architectures.⁸

Using the architect's workbench to evaluate architectures. With the workbench we normalize the effects of compiler optimization. The optimizer can be turned on or off, but all architectures receive the same degree of optimization. With a program trace we can view the effect of register allocation and create perfect allocation by reallocating after initial program execution.

The workbench was developed to allow top-down evaluation of a variety of architectures for a particular workload. The designer uses the workbench to select an instruction set customized to the particular application. The system is easy to use since in the initial evaluation only the basic instruction set parameters need to be specified. Unless explicitly added, the system will default to (assume that)

(1) All architectures have the same functional (ALU-type) operations and these operations correspond to the actions defined by the high-level source language

of the benchmarks (Pascal, for our benchmarks).

(2) All architectures have the same data paths (32 bits for this study).

(3) All instructions execute in unit time. We do not evaluate the effects of pipelining, but we do calculate the processor cycles spent in the data buffer and memory.

The system is designed to allow basic high-level tradeoffs, such as in instruction format selection, instruction encoding, register-set size and organization, and cache size and organization. After making an initial evaluation to select several promising candidate architectures, the designer would supply the additional information specifying ALU vocabulary and timing, pipeline timing templates, pipeline interlocks, etc., for a complete behavioral simulation.

Currently our system does not include facilities for pipeline timing evaluation, although it is being extended to include all of the above mentioned features.

For this article we based our tradeoff measurements on memory traffic; they do not directly include pipeline-cycle counts. Where differences arise, we will estimate the effect on cycle count.

An instruction set consists of a tradeoff between memory bandwidth and processor storage, as well as processor-decode requirements (see Figure 2). There are

several basic tradeoffs based on instruction encoding and available processor storage. The amount of complexity associated with instruction decode (the conciseness of the encoding) determines both the number of instructions fetched and the number of bits required from memory to interpret a program. The number of registers available for the allocator, together with the allocation strategy, determine the data traffic. Both instruction and data references to memory can be diminished by the presence of cache, either separate caches for the instruction and data stream or an integrated cache for both.

If we use care in evaluating relatively close architectural alternatives, we can get a good idea as to which architectural strategies are better than others, even though we may not be able to determine an exact optimum. In dealing with two similar designs, we can invoke a principle of marginal utility: Enhance a base design by an alternative that provides the maximum performance per unit cost. In order to use this in our analysis, we make the assumption that all processor variants under study have the same operational instruction set—they execute the same data transformational instructions (add, shift, etc.), even though they may differ in architectural instructions required by their instruction set (number of loads and stores in a register architecture, or number of pushes and pops in a stack architecture). By

Table 1. Benchmark sizes for stack architecture.

Benchmark	Static Size (bytes)	% Actually Referenced	Dynamic Size (bytes)	Instructions (Dynamic)
CCAL	12,980	63%	4,391,864	1,058,262
Compare	8,948	60%	35,113,324	8,538,373
PCOMP	71,276	69%	22,084,724	5,323,939
PASM	15,424	80%	17,814,260	4,352,798
Macro	73,980	53%	2,538,512	617,765

Table 2. Four instruction sets studied.

Type	Formats	Encoding	Addressability
Fix32	Two basic types: (1) Load R_1 , with Mem [addr] or (2) $R_1 \leftarrow R_2$ op R_3 (all operands in registers)	All instructions occupy 32 ^b	Word (32 ^b)
OBI360	As with Fix32 plus RX type,* $R_1 \leftarrow R_1$ op Mem [Addr]	Memory refr instr use 32 ^b ; register refr instr use 16 ^b and 32 ^b	Half word (16 ^b)
Stack	Stack formats	Instructions with op- code only 8 ^b ; mem- ory refr instr use 32 ^b	Byte
B6700	Stack formats with special encoding of constants and pointer-referenced memory	Instr with opcode only 8 ^b ; memory refr instr use 16 ^b	Byte

*Register set machine operations take two independent source operands (R_1 and R_2) and place the result in either an independent register (R_3) or one of the source operands (say, R_1). Most RISC machines as well as our Fix32 use the former convention, while the IBM System/360 uses the latter. For our code generator there is little advantage for the independent R_3 specification, i.e., the OBI360 data is approximately the same (within 1%) for either convention.

assuming the same operational (ALU) vocabulary, the same data path size, and the same arithmetic performance, we create a standard cost for that part of the processor. We are left with the comparison of only those parts of the architecture directly affected by the instruction set tradeoffs. These include the decoder, the register set size, instruction cache size, and data cache size. We assume that small tradeoffs in register set size and decoder size will not materially influence the cycle time itself. We will comment on this assumption later before drawing general conclusions.

The result of enhancing a base design with various alternatives provides insight into a *local* optimum; it does not find a *global* optimum. While we are able to comment on RISC and register-set-based instruction variations, we will not comment here on significantly different instruction sets (such as complex stack machines).

The benchmarks. We selected the benchmarks used in this study as representative of workstation applications. They consist of five Pascal programs originally used by Alpert.⁹ Some static and dynamic

measures for the benchmarks are given in Table 1. All data is for a stack machine with fixed 32-bit instructions (similar to P-code). The static measure is the program size in bytes as compiled without linkage overhead; it includes only executable code and constants as allowed in the stack machine architecture.⁷ The percentage of code actually used, for the given input files, is between 53 percent and 80 percent. Since all stack instructions are 32 bits, the difference between dynamic size (divided by four) and instructions is the occurrence of pointers, especially those associated with procedure calls.

The CCAL benchmark emulates a desk calculator. It reads a script of calculations from a text file and produces results in another text file. As with the other benchmarks, any input files required are specified as part of the benchmark, thus defining a standard execution. The Compare benchmark compares two text files, producing a description of their differences (similar to the Unix Diff command). The PCOMP benchmark compiles a Pascal program by recursive descent and produces P-code output. The PASM benchmark assembles the P-code output from the P-code compiler. The Macro benchmark is a macro processor for the SCALD computer-aided design system.

The chosen benchmarks are representative Pascal programs of medium size. They represent program generation, file processing, and calculation. CCAL also represents an interactive (as opposed to batch) program, although it is driven from a script file to keep its execution standard.

Each of these benchmarks was executed once for each target architecture after analysis of its basic blocks. Subsequently the address trace of that execution was fed into the cache simulator. The results presented are for the mean of the (equally weighted) benchmarks.

RISC-CISC code analysis

In the following analysis we define the RISC-type architecture³ as follows:

- (1) Load-store architecture. It does not allow memory operands for ALU operations.
- (2) Register-file oriented.
- (3) Pipelined execution with short cycle time, delayed branch, and a few register-oriented instructions.
- (4) Fixed 32-bit instruction size.

The base RISC we simulate is called Fix32. It is a load-store architecture with a 32-bit fixed instruction size with a register-set size of 16. (We examine other register set sizes in later sections.) Because delayed branching can be applied to our base architecture and its extensions, the effect of delayed branching is not simulated. This does not influence the results, however, because delayed branching affects the extensions of our base architecture exactly the same way as it affects the base architecture itself.

While we present some data on a number of architectural possibilities (see Table 2), one is particularly interesting: OBI360 (Only-Binary IBM 360). This variation is generally similar to IBM System 360 with the storage-to-storage instruction format excluded. OBI360 makes two modifications to the RISC strategy:

1. It adds the "RX" format (32 bits), allowing one instruction operand to reside in memory:

$$R_1 := R_1 \text{ op Mem}[R_2 + \text{offset}]$$

2. It adds half-size instructions (16-bit) register-to-register instructions:

$$R_1 := R_1 \text{ op } R_2$$

Starting off with the minimum Fix32 architecture, we perturb this design in various ways, such as

- (1) by increasing the complexity of the instruction encoding (using OBI360 instruction formats), or
- (2) by increasing the number of registers available to the data stream.

We make the assumption that other things remain constant—the base instruction set operational vocabulary, cycle time (discussed later), etc. We first consider the effect of instruction set selection on memory traffic in the presence of various-sized caches, then we consider the issue of register set size, organization, and allocation policy on memory traffic, again in the presence of various-sized caches.

The instruction set

The instruction set itself is largely a compromise between the complexity of the decoder (and thus the ensuing size of the microcode, cycle-time, etc.) and the required memory traffic to support execution (and thus the number of memory references). The RISC approach has opted for minimum decode complexity and accepted a relatively high bandwidth requirement for the instruction execution.

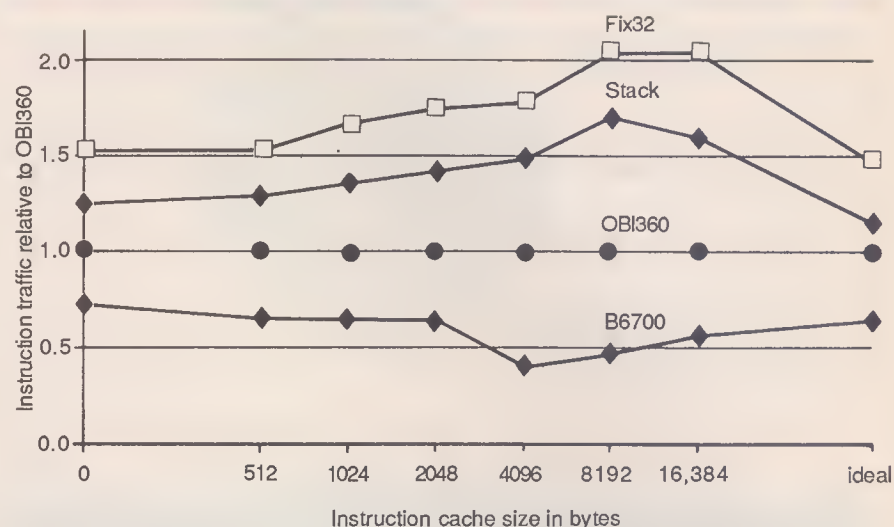


Figure 3. All architecture families (two-way associative 16-byte lines).

We can see the effects of instruction set selection on instruction traffic in Figure 3, which shows several instruction set families (see Table 2) relative to the memory traffic generated by OBI360 (with 16 general-purpose registers). The memory traffic is plotted for various cache sizes from no cache (zero bytes) to an infinite cache (ideal). A common cache policy of two-way associativity with a line size of 8 bytes has been selected across all caches and instruction sets. The relatively small line size tends to reduce the absolute amount of memory traffic required to support program execution and to diminish the difference among the instruction set families. Keeping the line size constant normalizes the cost for instruction caches for all of the families.

Figure 3 is interesting in a number of ways. Without a cache, the difference among architectures concerning the number of instruction bytes required to execute a program is about two to one (from the least dense architecture, the Fix32 with 16 registers, to the most dense architecture, the B6700).

Because an instruction cache benefits all architectures, it is difficult to see the relative benefit unless we normalize the traffic, as we have done in Figure 3. In the absence of a cache, the Fix32 architecture has 50 percent more instruction traffic than OBI360; this figure rises to 100 per-

cent for certain intermediate cache sizes (8K and 16K bytes). For these intermediate sizes, the OBI360 architecture has captured its working set, while the Fix32 has not yet done so. Ultimately, as all caches capture their working sets, the original relationship is restored, representing simply the number of references required to initially bring a program into the cache. Notice that for certain intermediate size caches, we can see relative differences of greater than five to one between the most dense and least dense architectures (B6700 to Fix32 in the 4K- to 8K-byte range).

Figure 4 shows the effect of marginal increases in the instruction decoder on overall dynamic program size and the resultant effect on cache performance. In Fix32-RX the RX format is added to Fix32. The RX format simply combines a load with a register-register operation. Since the RX format is the conjunction of two existing instruction types, its implementation cost can be minimal, depending on pipeline organization. About 10 percent fewer instructions will be executed with 10 percent fewer instruction-decode cycles. One might expect the pipelined program execution to improve by the same amount. It does not, because the RX instruction requires an extra cycle of interpretation, which may or may not be overlapped. Thus, in addition to a 10 percent reduction in instruction traffic, the

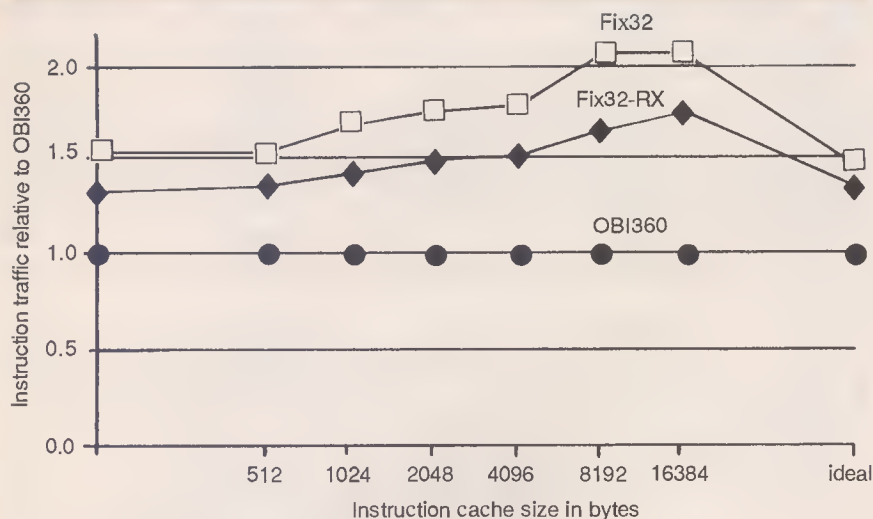


Figure 4. Memory referencing characteristics of register set architectures.

The effects of memory-to-register (RX) instructions

If no change is made to the pipeline, there is no reduction in the number of cycles executed, only reduction in instruction bandwidth. Basically, the sequence

```
Load R1, R2, Mem Disp; R1 ← Mem[[R2] + Mem Disp]
Add R3, R1           ; R3 ← R3 + R1
```

is replaced by

```
Add R3, R2, Mem Disp; R3 ← R3 + Mem[[R2] + Disp]
```

In both cases above, the sum is placed in R₃ in the same cycle. With the RX format, the pipeline is available one cycle early to admit a new instruction. However, if the pipeline uses the ALU for both address generation and execution, an RX instruction sequence will be limited to the same performance as before due to ALU contention.

If the processor is extended to include a separate adder for address generation and appropriate pipeline control, much (but not all) of this contention is avoided and performance enhancement can be realized.

As a rough estimate, suppose conditional branches represented 20 percent of instructions executed (and Fix32 load instructions, 30 percent). Now Fix32-RX reduces the frequency of load instructions to 20 percent (combining the load with an operation). However, the RX operation completes execution at the same time as the load-operate instruction pair. Thus, when the RX instruction immediately precedes a branch instruction that tests its result, no time is saved; otherwise a cycle is saved. Thus, if a branch is occupied every fifth position, a cycle is saved when the RX is located in any but the fourth position, or immediately preceding the branch. In the other three cases, the cycle is saved, giving a performance improvement potential (without register/ALU usage conflicts) of 7.5 percent rather than 10 percent.

number of execution cycles also reduces, but less than 10 percent (see the sidebar, "The effects of memory-to-register (RX) instructions").

Adding the RX format and a half-sized (16-bit) RR format to Fix32 results in OBI360 with one-third improvement in instruction bandwidth requirements for the no-cache case. The addition of half-size instructions to the format, however, does require additional decoder complexity to realize the alignment of instructions for proper decoding (see the sidebar, "The cost of half-size instructions"). Whether the two modifications to Fix32 to obtain OBI360 extend the cycle time we will discuss later.

OBI360 provides a striking improvement over Fix32 in the effectiveness of an instruction cache. Table 3 shows that OBI360 realizes the same miss rate as Fix32 with an instruction cache of exactly half the size of Fix32. Thus, by moving to OBI360 and adding decoder hardware, the resultant design would require fewer instruction cycles and would realize the same memory traffic with half of the Fix32 cache.

The data stream

What is the value of large register sets? What is the value of register windows? How effective is a data cache in the presence of a register set? These are some of the questions that the architect must address in creating a balanced instruction set design.

Registers have many roles. They hold: temporary values in expression evaluation, variables from statement to statement within a procedure, constants and pointers. Figure 5 shows types of references to data memory for variable and temporary accesses for source (Pascal and C) programs. If one had an architecture without registers (an all-memory architecture), 47 percent of the data references would be for temporary storage of intermediate results within the evaluation of expressions. The addition of two or three registers, whether through use of a stack or a register-instruction format, basically eliminates these references. When we ignore expression evaluation, the resultant traffic is for *extended source variables*—variables whose values are to be carried statement to statement within a procedure because of high probability of use.

The register allocator is responsible for

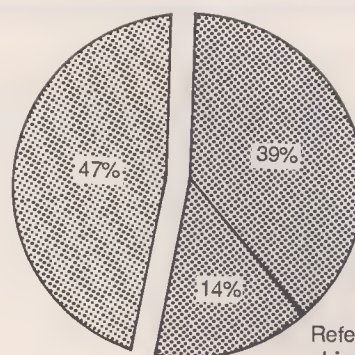
predicting the optimum assignment of variables to registers. Let us define this data traffic as the unity data traffic (thus excluding expression evaluation). Unity data traffic is not exactly the same as the variable traffic seen in a source program, however. The difference includes accesses for dynamic links, static links (the runtime organization), the fact that source variables may have arbitrary lengths but the physical system has a fixed-length bus (assumed to be 4 bytes), and finally that the compiler must create variables to control, for example, With and For statements.

Given different instruction sets with the same register set size and same compiler optimizer, the resultant data traffic will be the same for all instruction sets. How does data traffic compare to instruction traffic? Without data or instruction cache, the instruction traffic dominates the data traffic, but this quickly decreases when even a small instruction cache is added.

Figure 6 shows the instruction traffic for Fix32 and OBI360 instruction sets relative to unity data traffic. Both Fix32 and OBI360 include 16 general-purpose registers. For OBI360, the figure shows 10 percent more traffic for instructions than for data, while for Fix32, this difference is 60 percent. The addition of an instruction cache reduces the instruction traffic well below the unity data traffic for all architectures; thus, we need to rebalance the data traffic when considering a small instruction cache for the processor. The following four sections present different ways to rebalance the total memory traffic between instructions and data.

Single register set. Figure 7 shows the allocation of registers within a typical register set. Unity data traffic requires about eight registers in the set; they are allocated for constants, the evaluation stack, and state variables. The marginal value of adding eight registers to the initial eight depends upon register allocation. Figure 8 presents two allocation strategies: a very simple strategy wherein registers are allocated only among variables within a basic block, and a more elaborate strategy which allocates variables within a procedure. Global register allocation is by means of priority-based coloring. Notice that with or without optimization, the marginal value of more than eight additional registers is moot, and that a reduction from unity data traffic down to 0.65 can be achieved with straightforward global register allocation.

References to the evaluation stack



References to variables defined in the source program

Unity data traffic

References to implied data objects, such as compiler-created variables and runtime support

Figure 5. Distribution of data reference types.

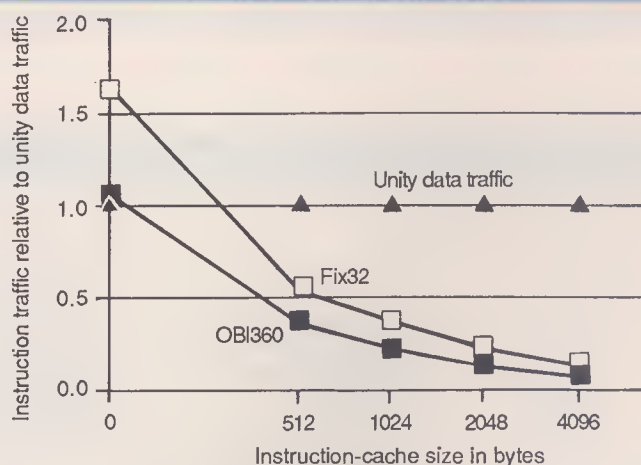


Figure 6. Instruction and data traffic as a function of architecture and instruction cache size.

The cost of half-size instructions

If a base processor design were intended to include 16^b instructions the following hardware might be added:

- (1) A two-word instruction buffer (IB) 2×32^b
- (2) Multiplexors from the IB into the instruction decoder
- (3) Finite-state machine for IB control
- (4) Modified program counter to allow both 16^b increment (for instruction control) and 32^b increment (for IB control)

Table 3. Relative traffic or miss rates. All traffic is relative to OBI360 without a cache and with 32^b -wide paths to memory.

For cache* size:	0	512	1K	2K	4K
Fix32	1.52	.44	.30	.19	.11
Fix32-RX	1.33	.37	.24	.17	.09
OBI360	1.0	.28	.17	.11	.05

*Two-way set-associative, 8-byte lines

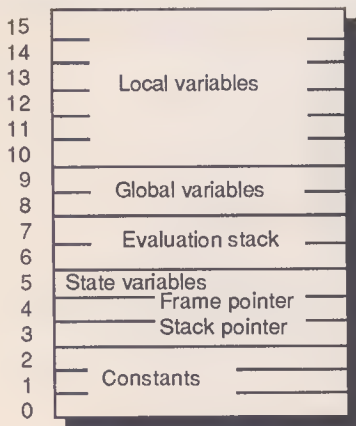


Figure 7. A possible register set usage outline.

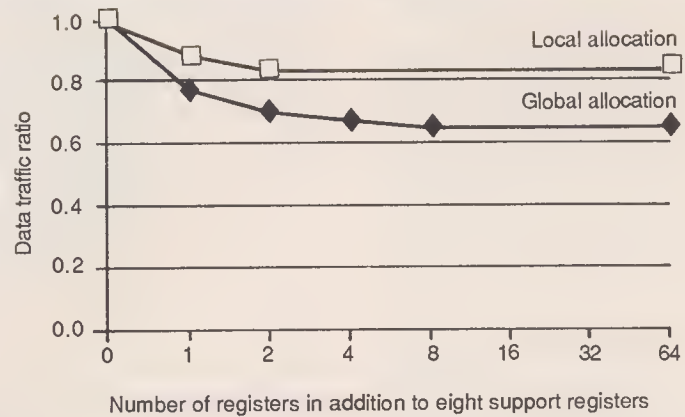


Figure 8. Single-register-set performance relative to unity data traffic.

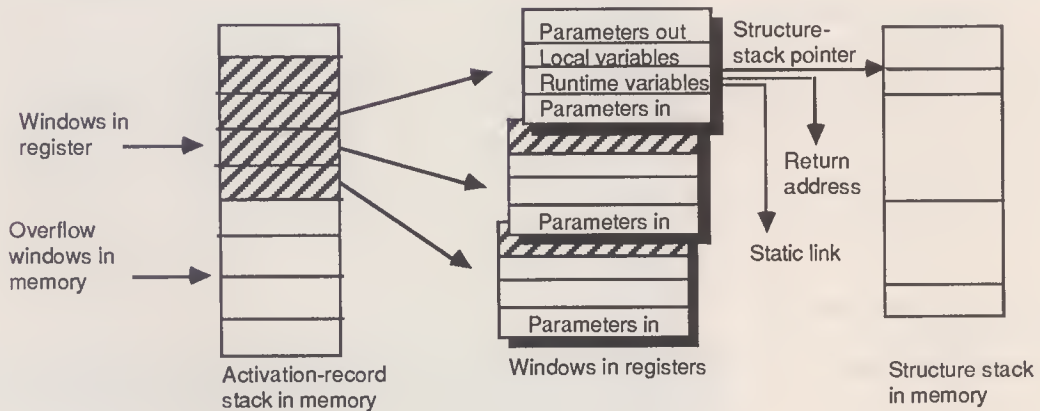


Figure 9. Multiple overlapping register windows.

To improve traffic beyond this requires allocation of registers across procedures. This can be done either in software, by interprocedural register allocation, or in hardware, through register windows.

Interprocedural register allocation. An interprocedural register allocator reduces the penalty of saving and restoring registers around procedure calls by allocating some registers as private to a particu-

lar procedure. Because procedures which mutually exclude each other from being in the call chain at the same time can share private registers, the number of registers required is reasonably small. Interprocedural register allocation has the additional advantage that runtime management variables, such as dynamic links, static links, and procedure-return addresses, can be allocated to registers using a method similar to that used for

procedure variables.

A simple and efficient interprocedural allocation scheme¹⁰ assigns private registers depth first in the call tree. Leaf procedures come first, callers of leaf procedures second, and so forth. This scheme treats recursion paths as single nodes and does not allocate private registers to procedures in such a path. Figure 10 shows the performance of an allocator based on this scheme, but extended

with an extra pass to detect variables potentially accessed through pointers¹¹; these variables cannot be allocated to private registers. The figure shows a traffic ratio of 0.5 with 8 registers and 0.4 with 32 registers available to the allocator.

Interprocedural allocation also has its drawbacks:

- procedures called recursively cannot have private registers,
- separately compiled modules must have knowledge of the register usage of all intra-module calls, or require register allocation during link time, and
- incremental compilation without an explicit linking stage such as Lisp environments can profit only partially from this scheme.

Interprocedural register allocation may turn single register sets into the most efficient local memory. Because of the drawbacks, however, unless explicitly stated otherwise we use only global (procedural) allocation in comparing different data buffers in the following sections.

Multiple overlapping register windows. Register windows allow a new set of registers to be made available for each procedure, with an overlap of registers between the caller and the called procedure to allow for passing of parameters. When a procedure call exhausts the number of windows, a window is freed by saving its data in memory. Whenever the data is needed again, it is restored from memory. The conditions which require a window save and restore are called overflow and underflow, respectively.

The hardware windows are organized as a circular buffer always covering the top part of the runtime stack. The outgoing parameters of the top window are the same as the incoming parameters of the bottom window. The hardware windows can be envisioned as rolling back (returns) and forth (calls) over the window stack in memory. Every activation record in the runtime stack has a fixed size, the window size. A second runtime stack, called the *structure stack*, keeps additional procedure variables which do not fit in the window stack. Every record in the window stack maintains a pointer to that part of the second stack which holds these variables.

Figure 9 shows the facets of a multiple-overlapping-window organization: the hardware windows covering the top of the window stack; the overlapping windows, holding parameters, local and runtime

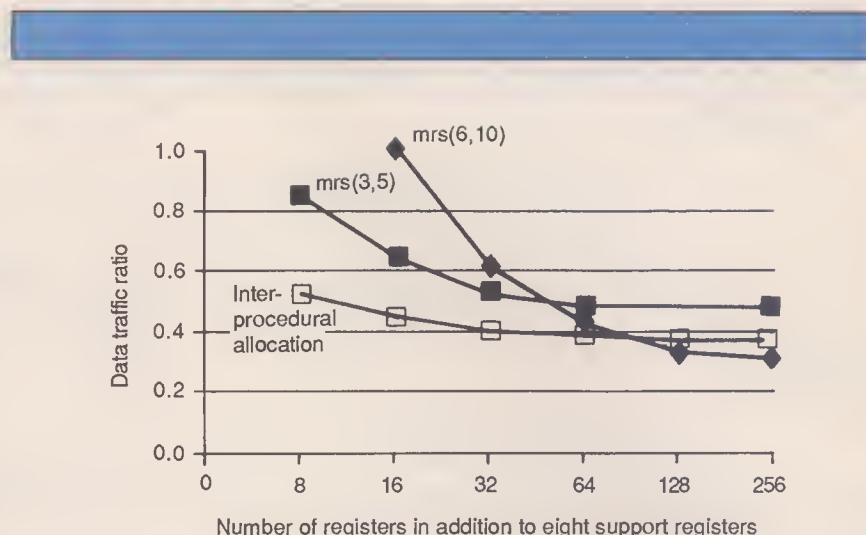


Figure 10. Performance of multiple overlapping register windows as a function of the total number of registers (relative to unity data traffic).

management variables; and the structure stack. An additional facility, which improves the performance of overlapping windows, allows pointer access to windows other than the one on top of the stack. This allows variables accessed through *var* parameters, the static-link, or pointers to be allocated in a window. For this article, we assume that overlapping windows include this facility.

We designate a window organization with N shared registers and M local registers by $mrs(N, M)$. The effectiveness of register windows depends mainly on two parameters: the size of the window and the number of windows. Figure 10 shows that a large window size, $mrs(6, 10)$, has a detrimental effect on performance for organizations with few windows. Few windows imply frequent under- and overflow conditions, and therefore a high penalty for large sets. However, a small window, $mrs(3, 5)$, has a negative effect on performance for an organization that has many windows. A small window captures fewer local variables and parameters, and therefore is less efficient compared with a large window when the over- and underflow traffic stops dominating total traffic.

To achieve the best of both large and small windows, global register allocation can be combined with small windows. This combination on the average outperforms both $mrs(3, 5)$ and $mrs(6, 10)$ for our benchmarks. In this article, however, we are concerned with the utility of large reg-

ister sets and especially those organizations actually implemented in general-purpose microprocessors. The buffer comparison in the following section, therefore, does not take small-window buffers into account. The performance of small-window buffers with and without register allocation and additional caching is presented elsewhere.¹¹

Single register set and cache combination. An argument in favor of larger register sets is that they will reduce the number of accesses to memory. Of course, a cache will do the same thing, and an interesting tradeoff occurs between increasing register set size and introducing or enlarging a data cache. Viewed from the memory system, enlarging either the register-set size or the cache size reduces the required memory traffic.

Using the traffic ratio as a function of provided storage for buffer comparison is not fair for two reasons: first, the storage provided does not accurately reflect the usage of chip area; and second, the traffic ratio does not completely determine processor performance.

To present buffer performance as a function of occupied area instead of the number of bits of storage, we define a simple storage-to-area mapping. The key points in this mapping are the inclusion of tag and status bits for caches, the distinction between the size of a cache RAM cell and a multiport register RAM cell, and the

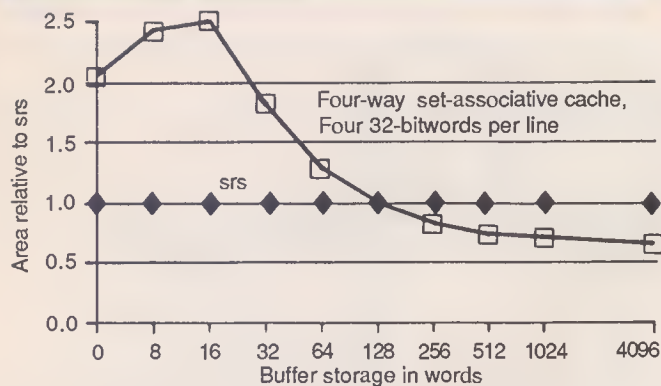


Figure 11. Buffer performance as a function of the amount of chip area.

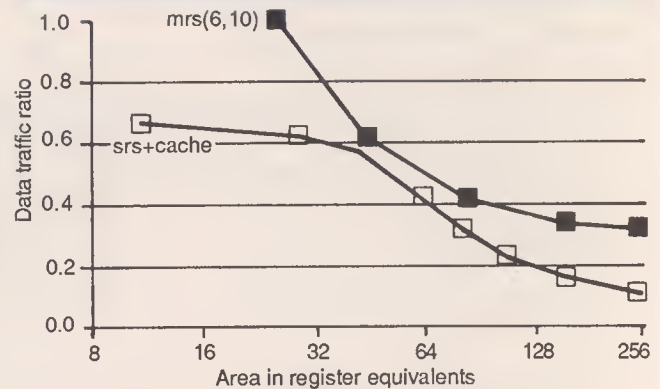


Figure 12. Data traffic ratio as a function of chip area.

inclusion of area for drivers, sense amplifiers, and tag comparators. The intent of this model is to penalize small register sets and caches for their inherent area overhead and to penalize register sets for their relatively large RAM cells, which they need to supply the high bandwidth required for pipelined processors. The main assumption underlying this model is that the RAM-cell size of a particular buffer is independent of the size of the buffer. Figure 11 shows the area of the cache* relative to the area of a register set as a function of the provided storage. The initial area disparity between the two buffers is mainly due to the tag comparators, and the state machine needed to control the cache. When the cache becomes larger than 32 lines, or 128 words, it actually takes less area than the register set, because the cache RAM cell is significantly smaller than the register RAM cell. Mulder gives a complete description and a validity assessment of the area model elsewhere.¹¹ The model parameters are all based on actual CMOS register set and cache designs.^{12,13}

The traffic ratio depicts the effectiveness of the data buffer viewed from the memory, but the situation is not quite the same when viewed from the processor. Access to a cache is usually a one- or two-

cycle operation, whereas access to a register set can be included within a cycle. Very large register sets may add a cycle or extend the cycle (see next section), but caches invariably add additional cycles to total program execution. A more accurate measure than the traffic ratio would be the ratio of the processor cycles spent in the buffer and main-memory combination and the cycles spent in the memory system without the presence of a buffer. Mulder describes this measure, the *cycle ratio*, in detail.¹¹

In a highly pipelined organization which keeps its buffer and memory system busy all the time, the cycle ratio describes the exact performance of the processor. Note that pipeline breaks, not caused by the memory and buffer system, may cause the real performance to deviate from the cycle ratio. Nonetheless, it is a good measure for evaluating different buffer organizations and their timing parameters.

Figure 12 summarizes the traffic ratio, and Figures 13 and 14 the cycle ratio of two buffering strategies. All three figures show their ratios as a function of occupied area in register equivalents; one register equivalent is the area occupied by 32 register bit cells. Figures 13 and 14 show the cycle ratio for a main memory access time of two and three cycles, respectively.* The figures show the ratios for a multiple-register-window organization, *mrs(6,10)*, and a

single-register-set and cache combination, *srs + cache*. The register allocator only uses four registers for allocation, but the architecture is assumed to have an additional eight as described before.

Both Figures 12 and 14 show a slight advantage for *srs + cache* over *mrs(6,10)* between 40 and 80 register equivalents. Before and after this interval, *srs + cache* performs significantly better than *mrs(6,10)*. Reducing the memory access time from three to two cycles, however, undoes the *srs + cache* advantage. Now *mrs(6,10)* is slightly better than *srs + cache* for the 40- to 80-register interval, and performs approximately the same for larger buffers. Nonetheless, an increase in average memory access time is always an advantage of the *srs + cache* organization. The case for multiple register sets with relatively large sets is slight, at best, and occurs only in the region of 32 to 128 registers for one- or two-cycle main memory access time. An important advantage for the *srs + cache* organization is its relative independence from the reference distribution. A smaller percentage of references to the window stack immediately degrades *mrs(6,10)* performance, while this is not necessarily the case for *srs + cache*.

Several buffer characteristics are not taken into account in this comparison because they lie outside the scope of the article or because insufficient data was available. These include the effect of system functions (interrupts, I/O, and context switches) and data-consistency

*The cache is four-way set associative with four one-word transfer units per line. The first data point, however, is a direct mapped one-line cache, and the second data point is a two-way set associative two-line cache.

*A cache hit takes one cycle, while a cache miss takes one cycle plus a memory access.

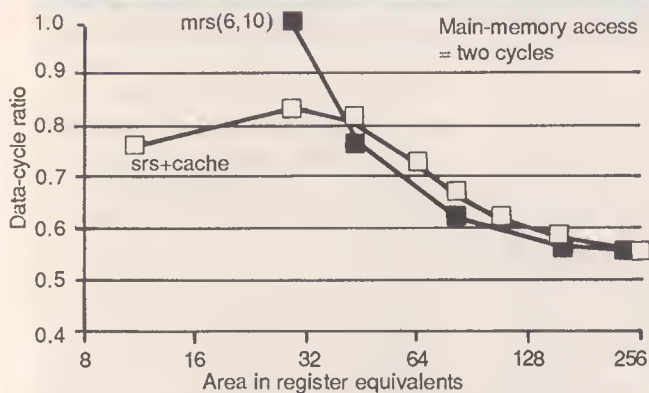


Figure 13. Cycle ratio as a function of chip area (two-cycle memory).

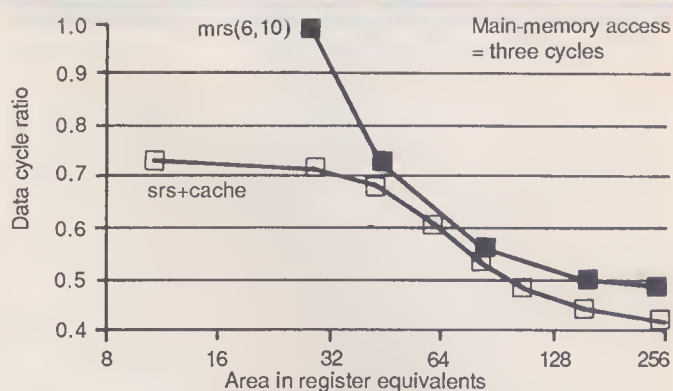


Figure 14. Cycle ratio as a function of chip area (three-cycle memory).

requirements in the case of multiprocessing.

Trading instruction traffic for data traffic

Memory traffic is the sum of the instruction traffic plus the data traffic. Assume that a processor with a Fix32 instruction set has eight general-purpose registers. Is it better to add registers, or to increase instruction complexity? Table 4 gives insight into these tradeoffs. Note that the table presents total traffic relative to the total instruction and data traffic of Fix32 with eight registers. If we increase the Fix32 register set size from eight registers to 16, we reduce the relative traffic from 1.00 to 0.76. However, if we increase the instruction complexity and retain a register set size of eight by moving from Fix32 to OBI360, the relative traffic also reduces from 1.00 to 0.76. Clearly it is desirable to do both, which would reduce the relative traffic to 0.56.

If we assume a Fix32 base design with a register set size of 16, there is almost no saving in increasing the size to 32 (unless coupled with interprocedural register allocation). The instruction traffic and the data traffic remain essentially constant. By adding eight register windows, mrs(6,10), for a total of 128 registers, we reduce data traffic to 0.12 and instruction traffic to 0.52—a net savings of 0.12 references from Fix32 with 16 general-purpose

Table 4. Instruction and data traffic measured relative to the total traffic of our initial architecture, the Fix32 with eight registers.

Architecture	Register Set Size	I-traffic	D-traffic	Total
Fix32	8	0.68	0.32	1.00
Fix32	16	0.55	0.21	0.76
Fix32	32	0.55	0.21	0.76
Fix32-MRS	128	0.52	0.12	0.64
OBI360	8	0.44	0.32	0.76
OBI360	16	0.35	0.21	0.56
OBI360	32	0.35	0.21	0.56
OBI360-MRS	128	0.32	0.12	0.44

registers. On the other hand, if we simply retain the 16 general-purpose registers and modify the instruction set from Fix32 to OBI360, we realize a savings of 0.20 references. Thus, the addition of 112 registers and window control results in only 60 percent of the traffic reduction we can achieve with better instruction encoding (the change from Fix32 to OBI360).

Cycles and cycle time

So far we have dealt with design alternatives assuming that the processor execution (independent of memory traffic) was

unaffected. In this section we review the alternatives examined and assess the possible impact on either the number of cycles required to execute a program, or the cycle time itself.

The cycle count. Concerning the cycle count, consider the principal design alternatives presented.

(1) We first considered Fix32 compared with Fix32 plus a memory-to-register instruction format (Fix32-RX). This change will either leave cycle count unaffected or it will allow a modest (less than 10 percent) decrease in the number of cycles (see the sidebar, "The effects of memory-to-register (RX) instructions")

when hardware is added to the pipeline.

(2) We next considered the change from Fix32-RX to OBI360. This change adds the I6^b (or "half size") instruction; it should have no effect on cycle count (also see the sidebar, "The cost of half-size instructions").

(3) Finally we considered adding either additional registers or a data cache to a I6-register design. Additional registers may increase cycle time (discussed below), while a cache may require either more or fewer cycles than a large register set, depending on the memory access time.

Without interprocedural allocation, there is obviously no marginal utility, in terms of traffic ratio or cycle count, in enlarging a single register set beyond 16 registers. With interprocedural register allocation, we see an improvement in memory traffic by increasing the total register set to 32, after which again there is little or no marginal utility. The cycle count will decrease as memory traffic decreases.

Part of the effect on cycle count of moving from a base I6-register design to multiple register set versus the move to register set and cache combination was discussed in "The data stream" and shown in Figures I3 and I4. The design with a cache clearly expends fewer cycles accessing objects not in the register set. On the other hand, when we call a new procedure, the multiple register set has an advantage because it requires fewer cycles to save and restore register values. However, many parameters are involved (frequency of calls, number of parameters, cache miss penalty, etc.), and a specific situation may find one approach significantly superior.

The cycle time. Concerning cycle time, we have considered two design extensions that could adversely affect the internal processor cycle time:

(1) The extension of Fix32 to OBI360 increases the instruction decoder complexity.

(2) Increasing the number of registers may increase register access time.

Cycle time is typically determined by one of four paths:

- (1) Instruction decode
- (2) Register access
- (3) Cache access
- (4) ALU operation and condition-code set

The designer usually is confronted with physical limitations which determine the longest path from one of the above considerations. Once that is determined, the

other paths can be increased to roughly the same duration so as to minimize costs. A review of several recent RISC designs (RISC II¹⁴ and SOAR¹⁵) indicates that register access time has been a primary determinant of cycle time, not instruction decode. In MIPS,¹⁶ a pipelined RISC with I6 general-purpose registers, the main cycle time determinant was pipeline and exception control circuitry. It seems in some cases that the use of smaller register sets could improve the cycle time, while the use of OBI360 should have a negligible effect on cycle time.

Other considerations in instruction set design

So far in this article, we have examined such aspects of performance as the basic criteria in instruction set selection and design. Many important functional considerations are not performance related, or are related to performance only in a secondary way. In this section we briefly remind the reader of several of these considerations.

Compatibility. The primary level of transportability of programs is the instruction set, not the higher-level language. The reason that the VAX series from Digital Equipment Corporation and the 68000 series from Motorola Corporation resemble their antecedents is not simply an issue of designer's preference, but of customer preference. The VAX instruction set is a good case in point. It is derived from (built upon) the PDP-II, a I6-bit architecture noted for its flexible use of addressing modes. Rather than abandon the addressing modes, the VAX designers enhanced them, preserving subset compatibility with the PDP-II yet achieving generous functionality in the 32-bit arena. The resultant design provides for relatively concise encoding of programs. Unfortunately, the flexible object identification introduces extra cycles into instruction interpretation and makes instruction pipelining more difficult. Still, the VAX series has become an industry standard because of compatibility, connectivity (I/O capabilities), and availability of software, not simply performance.

Technology. In the context of microprocessors, chip area constraints force a design to fit in a fixed area. Performance may be severely limited where an instruction set has been selected without consideration of these area constraints. Pins as well as area may constrain designs, restricting access to memory. This provides a premium for those architectures which make the best use of memory bandwidth.

Most instruction sets are children of their technological times. Assumptions concerning memory size, access time, and the relative cost of registers, as well as the ability of a compiler to use those registers, determine the tradeoffs that go into a resultant instruction set specification.

Life cycle costs. Compatibility, technology, performance, software and hardware development effort, maintenance costs, and hardware reliability are ingredients in determining the total system cost (and profitability) for the product over its lifetime. Life cycle cost is clearly an ultimate measure, and our performance discussion is simply one factor of it.

While instruction set design involves many considerations, performance data is an important component. Indeed, performance may well be a primary consideration in certain custom application-specific designs.

In evaluating design tradeoffs, the architecture-simulation platform described above provides valuable relative performance data and greatly assists in optimizing design choices from the top down. The workbench provides an early assessment of the relative merits of different design options and facilitates the selection of the option with the greatest marginal utility.

In this article we limited the data presented to five benchmarks selected from a Pascal-type environment similar to that reported on by other researchers in the area. Different environments might produce significantly different results.

The principal design alternatives we have examined are Fix32 and OBI360:

(1) Fix32 is RISC-like in formats and encoding, but *does not reduce* the ALU instruction vocabulary. The functional instruction set is the same for all architectures studied, so as to keep the ALU cost relatively constant. RISC implementation with fewer instruction types may require additional instruction memory traffic and exaggerate the difference between

architectures (see Figure 3).

(2) OBI360 is Fix32 with the addition of both the RX and half-size instruction formats. OBI360 is not System 360 or System 370. Typical S/370 code includes the operating system interface, calling conventions, and runtime prolog/epilog, which significantly distorts the locality and cache results presented here. However, to evaluate System 370 code is to evaluate an evolution of software, not simply instruction set technology. The relatively good results of OBI360, however, are a compliment to the basic tradeoffs made by System 360 instruction set designers over twenty years ago.

Concerning the various alternatives:

(1) Fix32 (a simple load-store architecture) with 16 or fewer registers and *without* cache represents a reasonable design point for a minimum cost processor. Note that the minimum processor cost was achieved with a 50 percent to 100 percent increase in instruction traffic (Figure 3), compared with more complex designs.

(2) If Fix32 is extended to reduce memory traffic, instruction encoding and formats should receive priority attention over the expansion of an instruction cache. By adding modest decode hardware to Fix32, we create OBI360, which achieves the same memory performance as Fix32, but uses an instruction cache of only half the Fix32 cache size (Table 3).

(3) From data traffic considerations alone, it seems that OBI360 with a register set of about size 16 plus a small data cache is preferable to multiple register sets for most area combinations (Figures 12 and 13).

More generally, instruction set designers cannot afford to ignore issues of code density in favor of instruction simplicity or decoding ease. Instruction bandwidth can be a significant component of memory traffic and, ultimately, processor performance. Using larger register sets to reduce data traffic from memory makes sense only when efficient instruction encoding is used to make a corresponding reduction in instruction traffic. *Balanced optimization* is the key to overall instruction set efficiency. □

Acknowledgments

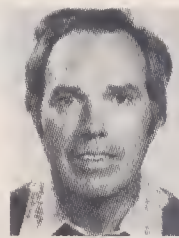
Several basic ideas used in the computer architecture simulator were based on the work of our colleague Don Alpert. We also used his benchmark programs. The U-code system included in our platform was the work of John

Hennessy and Gio Wiederhold and their students here at Stanford.

This research was supported by NASA under contracts NAG2 248 and NAGW 419, and by an IBM graduate fellowship.

References

1. Also see product announcements for Fairchild Clipper, Hewlett-Packard Spectrum, Advanced Micro Devices AM29000, and MIPS Corp.
2. Mark Hill et al., "Design Decisions in SPUR," *Computer*, Nov. 1986, pp. 8-22.
3. John Hennessy, "VLSI Processor Architecture," *IEEE Trans. Computers*, Dec. 1984, pp. 1221-1246.
4. David A. Patterson and Carlo H. Sequin, "RISC I: A Reduced Instruction Set VLSI Computer," *Proc. 8th Ann. Symp. Computer Architecture*, May 1981, pp. 443-458.
5. R.P. Colwell et al., "Computers, Complexity, and Controversy," *Computer*, Sept. 1985, pp. 8-19.
6. Michael J. Flynn, "Towards Better Instruction Sets," *Proc. 16th Ann. Microprogramming Workshop*, Oct. 1983, IEEE Computer Society Press, pp. 3-8.
7. Chad L. Mitchell, *Processor Architecture and Cache Performance*, Tech. Report CSL-TR-86-296, Computer Systems Laboratory, June 1986.
8. Scott Wakefield, *Studies in Execution Architectures*, PhD thesis, Stanford University, Jan. 1983.
9. Donald Alpert, *Memory Hierarchies for Directly Executed Language Microprocessors*, PhD thesis, Stanford University, June 1984.
10. Peter A. Steenkiste, *LISP on a Reduced-Instruction-Set Processor: Characterization and Optimization*, PhD thesis, Stanford University, Mar. 1987.
11. Johannes M. Mulder, *Tradeoffs in Data-Buffer and Processor-Architecture Design*, PhD thesis, Stanford University, 1987. In preparation.
12. M. Horowitz and P. Chow, "The MIPS-X Microprocessor," *Proc. Wescon 1985*, Stanford University, 1985.
13. Anant Agarwal et al., "On-chip Instruction Caches for High Performance Processors," *Proc. Advanced Research in VLSI*, Stanford University, Mar. 1987.
14. M.G.H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI*, PhD thesis, UC Berkeley, Oct. 1983.
15. David Ungar et al., "Architecture of SOAR: Smalltalk on a RISC," *11th Ann. Symp. Computer Architecture*, Ann Arbor, Mich., June 1984, pp. 188-197.
16. Steven A. Przybylski et al., *Organization and VLSI Implementation of MIPS*, Tech. Report 84-259, Computer Systems Laboratory, Apr. 1984.



Michael J. Flynn is a professor of electrical engineering at Stanford University, where he served as director of the Computer Systems Laboratory from 1977 to 1983. He is also a senior consultant at Palyn Associates, a computer design firm in San Jose, California. He was a cofounder and vice president of Palyn in 1973 while on leave from Johns Hopkins University.

Flynn worked at IBM for ten years in the areas of computer organization and design. He was design manager of prototype versions of the IBM 7090 and 7094/II, and later for the System 360 Model 91 CPU.

Flynn received his PhD from Purdue University in 1961.



Chad L. Mitchell is currently Chief Technical Officer at Great Wave Software, which he helped found in 1984. He has written several programs for personal computers, including the award-winning Concertware music system.

Mitchell received a BA in mathematics in 1979 and a BS in computer science in 1980 from the University of Utah, both Magna Cum Laude. He received an MS in computer engineering in 1982 and a PhD in computer science in 1986 from Stanford University. While there, he coauthored the Gambit prototyping language.



Johannes M. Mulder is currently completing the requirements for the PhD degree at Stanford University. His primary research interests are in computer architecture, VLSI, and computer and memory system design and evaluation.

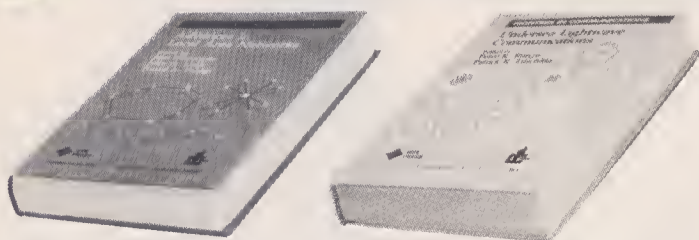
Mulder received the MS degree in electrical engineering in 1982 from Delft University of Technology in the Netherlands.

Readers may write to Flynn at Electrical Engineering Dept., ERL 452, Stanford University, Stanford, CA 94305-4055.



From the IEEE Communications Society in conjunction with the IEEE PRESS . . . INTRODUCING

FRONTIERS IN COMMUNICATIONS



Advances in Local Area Networks

Edited by Karl Kümmerle, IBM Research Laboratory, Zurich, Switzerland; John O. Limb, Hewlett Packard Laboratories, Great Britain; and Fouad A. Tobagi, Stanford University. 1987. 616 pages. ISBN-0-87942-217-3.

Local Area Networks (LANs) are evolving rapidly and can take many forms. First generation LANs operating at speeds up to 10 Mb/s are widely deployed in office environments today. Second generation LANs and Metropolitan Area Networks (MANs) operating at about 100 Mb/s are the subject of current standardization efforts and third generation LANs operating in the Gb/s region are the object of current research.

The editors of this book point out that Local Area Networks differ from classical communications channels, in that firstly, the transmission medium is shared by a number of users and, secondly, that at least some of the information is sent in packets containing a destination address. These two properties are also shared by many MANs. LANs and MANs exploit similar concepts, the primary difference being in the geographical coverage provided.

The book provides a description of the most important principles of LANs, and emphasizes the exciting new directions LAN technology is taking.

Contents: Introduction; Evolution of Local Area Networks; Fiber Optics Applied to LANs; Performance; Integrated Traffic; Bridges and Gateways; Software and Applications; Subject Index.

Clothbound Order No. PC02105 List \$56.50 Member \$42.40

Undersea Lightwave Communications

Edited by Peter K. Runge and Patrick R. Trischitta, AT&T Bell Laboratories. 1986. 644 pages. ISBN-0-87942-201-7.

Lightwave systems are being installed on land at a phenomenal pace. Initial considerations might suggest that it would be a relatively easy task to apply terrestrial optical technology to crisscross the world's oceans and seas. This is far from true. The technology necessary for an undersea lightwave communication system is very different from that for terrestrial systems and is very complex and demanding.

The book provides a comprehensive description of the latest worldwide progress in undersea lightwave technology for transoceanic communications systems. It also offers an outlook on future technology options.

Contents: Background; Undersea Lightwave Systems; Undersea Fiber; Undersea Fiber Cables; Undersea Lightwave Repeater Design; Undersea Electrical Components; Undersea Optical Devices; Assuring the Reliability of Undersea Electrical and Optical Devices; Supervisory, Control, and Terminal Equipment; Future Undersea Lightwave Systems; Author and Subject Indexes.

Clothbound Order No. PC01933 List \$60.80 Member \$45.60

Focused and comprehensive, our new book series provides state-of-the-art coverage on subjects of major importance to professionals in communications and related fields.

Frontiers is a manifestation of IEEE ComSoc's commitment to technical and educational excellence. It is our intention to make each title in the series a most definitive and up-to-date source on its subject. Each *Frontiers* volume is focused on a timely and important topic not fully covered in the literature. Each is designed to serve the information needs of the interdisciplinary audience involved in modern communications.

Definitive sourcebooks for the practicing professional

LANs . . . ISDNs . . . packetized voice and data . . . military communications . . . digital radio . . . regardless of subject—each *Frontiers* volume is a complete, coherent, and up-to-date sourcebook featuring . . .

- a comprehensive introduction to the subject
 - lead-in tutorials to aid comprehension and place materials into perspective
 - invited papers by "world-class" authors
 - bridging and other materials supplied by expert editors
 - extensive references and cross-indexing
- Thus, each subject comes "full-circle" in a unique, heavily illustrated and handsomely-formatted book.

The first two books in the series are described on this page. We invite you to place your order today and see for yourself why we are so enthusiastic about *Frontiers in Communications*. A convenient order form is provided.

Amos E. Joel, Jr. is Editor of the *Frontiers in Communications* series. A past President of the IEEE Communications Society and a Fellow of the IEEE, Mr. Joel is a widely published and world-renowned authority on communications switching.

Books in the Frontiers series feature an easy-to-read type face and numerous charts, diagrams, photographs and tables. The first two books contain nearly 800 illustrations. Page size is 17cm. by 25cm.

Please send me the *Frontiers in Communications Series* books I have entered below. (Note: Member price applies to first copy only of each title. Additional copies of the same title are at list price.)

Advances in Local Area Networks Quantity Amount
(PC02105) List \$56.50; _____ \$
IEEE members \$42.40 _____ \$

Undersea Lightwave Communications _____ \$
(PC01933) List \$60.80; _____ \$
IEEE members \$45.60 _____ \$

Subtotal _____ \$

Postage/Handling Charge* _____ \$

N.J. Residents add 6% state sales tax _____ \$

Total Amount _____ \$

*Postage/Handling Charge: For orders totaling \$1.00 to \$50.00, add \$4.00; \$50.01 to \$75.00, add \$5.00; \$75.01 to \$100.00, add \$6.00; \$100.01 to \$200.00, add \$8.00; over \$200.00, add \$15.00.

CHECK APPROPRIATE BOX:

☐ Payment enclosed. (Check payable to IEEE in U.S. dollars drawn on a U.S. bank.)

☐ Bill me. Purchase Order No. _____

☐ Charge to my credit card:

☐ MasterCard ☐ American Express ☐ Diners Club ☐ VISA

Card No. _____ Exp. Date _____

Signature _____

Name (print) _____

IEEE number _____

Address _____

City/State _____ Postal Code _____

Return this coupon to: IEEE Service Center, 445 Hoes Lane

P.O. Box 1331, Piscataway, NJ 08855-1331

Credit card orders: (201) 981-0060 Ext. 350

Invoice orders: (201) 981-1393

FC-D

An Overview of Some Formal Methods for Program Design

C.A.R. Hoare

Oxford University Computing Laboratory

The code of a computer program is a formal text, describing precisely the actions of a computer executing that program. As in other branches of engineering, the progress of its implementation as well as its eventual quality can be promoted by additional design documents, formalized before starting to write the final code. These preliminary documents may be expressed in a variety of notations suitable for different purposes at different stages of a project, from capture of requirements through design and implementation, to delivery and long-term maintenance. These notations are derived from mathematics, and include algebra, logic, functions, and procedures. The connection between the notations is provided by mathematical calculation and proof.

This article introduces and illustrates a selection of formal methods by means of a single recurring example, the design of a program to compute the greatest common divisor of two positive numbers. It is hoped that some of the conclusions drawn from analysis of this simple example will apply with even greater force to software engineering projects on a more realistic scale.

Requirements

Imagine that a software engineer is called upon to construct a mechanism or

The design of a small program, like that of a large system, requires a variety of formal methods and notations, related by mathematical reasoning.

subroutine to compute the greatest common divisor z of two positive integers x and y . By an even greater feat of imagination, assume no prior knowledge of the concept of greatest common divisor, or of how to compute it. So the first task is to ensure that the engineer and client have the same understanding of what is required. Let us suppose first that they agree to confine attention to positive whole numbers (excluding zero). The required relationship between the parameters (x, y) and the result (z) may be formalized as follows:

- D1.1 z divides x
- D1.2 z divides y
- D1.3 z is the greatest of the set of num-

bers satisfying both these conditions
D1.4 " p divides q " means "there exists a positive whole number w such that $pw = q$ "

D1.5 " p is the greatest member of a set S " means " p is in S , and no member of S is strictly greater than p "

It is essential that the notations used for formalization of requirements should be mathematically meaningful, but it would be unwise to place any other restriction upon them. Even in this simple example we have used logic, arithmetic, and set theory. An engineer should take advantage of this early notational freedom to ensure that the formalization of requirements is simple and well-structured so that, together with its informal explanation, it obviously describes what is really wanted, and not something else. There can never be any formal method of checking this vital correspondence, because there should not be any more obviously correct description to check it against. Clarity is our only defense against the embarrassment felt on completion of a large project when it is discovered that the wrong problem has been solved.

The most important mathematical notations needed in requirements specification are also the simplest—for example, the familiar Boolean connectives AND, OR, and NOT. The connective AND (known as *conjunction*) is needed to put together the separate requirements imposed upon a product. In the example above, we want

the answer z to be a divisor of x and we want it to be a divisor of y . Sometimes we need universal quantification ("for all x "), which is a generalization of AND. The connective OR (known as *disjunction*) is needed to achieve simplicity and symmetry by maintaining a high level of abstraction. Sometimes we need *existential quantification* ("there exists"), which is a generalization of OR. For example, in clause D1.4 above, we want there to exist a w exactly dividing x , but we don't care if there is more than one, and if so we don't care which one is chosen. And we certainly don't want to describe in gory detail how a computer is to find one. Finally, it is often easier and clearer to say what we want the product not to do. For example there must be no divisor of x and y greater than z . In the formulation of requirements, we certainly need the simple Boolean connectives AND and OR, and especially NOT.

By using such a powerful specification language, we run the risk of writing a falsehood, inconsistency, or other absurdity which could never be implemented. This risk can be eliminated by an early consistency check. In our example, what we need to check is that for every pair of positive numbers x and y there exists a number z with the properties specified in D1.3. A proof of this has three steps:

P1.1 The number one is a divisor of every number. So it is a common divisor of every pair of numbers. This shows that the set of common divisors of two numbers is non-empty.

P1.2 Each number is its own greatest divisor, so every set of divisors is finite. The common subset of any two finite sets is also finite. So the set of common divisors of two numbers is both finite and non-empty.

P1.3 Every finite non-empty set of integers has a greatest member. So the maximum used to define the greatest common divisor always exists.

At the end of requirements analysis, we aim to have a mathematically precise, complete, consistent, and above all obviously appropriate mathematical description of the result we want. It would be very nice to input this description into some suitably programmed computer, and get the computer to translate it automatically into a subroutine. On each call of the subroutine, it would return the greatest common divisor of any pair of numbers supplied as arguments. If we are willing to overlook the problem of efficiency, there

In the formulation of requirements, we need the simple Boolean connectives AND and OR, and especially NOT.

is an easy way of doing this, known as the British Museum algorithm. A computer can be programmed to start with some standard set of axioms of mathematics, and to use them to generate at random all provable mathematical theorems. It will therefore eventually generate such startling theorems as

One is the greatest of all numbers which divide both two and three.

So if we want to know the greatest common divisor of two and three, all we need to do is to program another computer to recognize when the British Museum computer has produced a result of this form. Then we just wait for it to do so. We may have to wait a very long time—under reasonable assumptions, all the particles in the universe will decay to photons before our fastest computers can carry out even the most trivial calculations. And if the theorem-generator attempts a strategy much better than random generation, it is difficult to avoid the risk that some true theorem will never be generated. In mathematics, the question of efficiency is rightly considered irrelevant, but in using computers to do mathematics, efficiency cannot forever be ignored—or else it will really be forever. Efficiency is therefore the main driving force in the development of an acceptable program to meet its mathematical specification, as shown in the following sections.

Logic program

A solution to the unavoidable problem of efficiency is to recast the specification of requirements into some notational framework less powerful and less general than the whole of mathematics. This restricted notation is so designed that its use will be rewarded by more efficient execution on a computer. One of the concepts

of mathematics that is most difficult to implement effectively is negation of specification. For example, if you want a program that does not trigger a holocaust, you cannot hope to write a program that does trigger a holocaust and just negate it before submitting it to the computer. For this reason, even the highest level logic programming languages prohibit or severely restrict the use of negation, requiring the programmer to implement it whenever needed.

Here is an idealized logic program to compute the greatest common divisor of two positive integers. To help in checking its correctness, it has been designed to preserve as far as possible the structure and clarity of the original requirements. We assume that "isproduct" and "differsfrom" are available as built-in predicates on positive integers.

L2.1 $\text{isdivisor}(x, z)$ if there exists a w not greater than x such that $\text{isproduct}(z, w, x)$

L2.2 $\text{iscommondiv}(x, y, z)$ if $\text{isdivisor}(x, z)$ and $\text{isdivisor}(y, z)$

L2.3 $\text{isgcd}(x, y, z)$ if $\text{iscommondiv}(x, y, z)$ and for all w from z to x $\text{isnotcommondiv}(x, y, w)$

L2.4 $\text{isnotcommondiv}(x, y, z)$ if $\text{isnotdiv}(x, z)$ or $\text{isnotdiv}(y, z)$

L2.5 $\text{isnotdiv}(x, z)$ if for all w from 1 to x $\text{isnotproduct}(z, w, x)$

L2.6 $\text{isnotproduct}(z, w, x)$ if $\text{isproduct}(z, w, y)$ and $\text{differsfrom}(y, x)$

This program is a great deal more complicated than the requirements specification in the previous section. The obvious reason is that the absence of negation in the programming language requires explicit programming of a search through all possibilities before a negative answer is given. In order to ensure termination, a finite range for each search has to be specified, and setting this limit requires knowledge of the application domain. For example, in L2.3 we rely on the fact that the common divisor of two numbers cannot exceed either number.

Note that in the best known programming language, Prolog, it would be permitted to replace L2.3 to L2.6 by the single clause

L2.3' $\text{isgcd}(x, y, z)$ if $\text{iscommondiv}(x, y, z)$ and $\text{not}(\text{iscommondiv}(x, y, w))$ and $\text{isgreater}(w, z)$

This restores the brevity of the specification of requirements, but when submitted to a standard implementation of Prolog, this program may turn out to be slightly worse than the British Museum

algorithm because it does not terminate at all. The trouble is that the "not" of Prolog does not mean the same as the NOT of mathematics, logic, or even normal technical discourse, and its meaning cannot be fully understood except in terms of the way that Prolog programs are executed. In fact, in Prolog the "and" and the "or" also have peculiar meanings, as a result of which they are not even symmetric. The motive for this was to achieve greater efficiency of execution, particularly on traditional sequential computers.

In spite of considerable ingenuity of implementation, logic programs are inherently inefficient, particularly if the specification takes proper advantage of the power of the available combination of conjunction and disjunction. The reason is that in principle all combinations of possibilities described by each disjunction have to be explored, in order to be sure of finding one that is consistent with the rest of the specification. Consider the conjunction of $(A \text{ or } B)$ with $(C \text{ or } D)$. This gives rise to four alternatives, $(A \text{ and } C)$ or $(A \text{ and } D)$ or $(B \text{ and } C)$ or $(B \text{ and } D)$, all of which may have to be tried. An existential quantifier multiplies the number of cases by a larger factor, and if recursion is involved, the number of cases to be explored increases exponentially. All but one of these cases will eventually be discarded.

Algebra

A possible way of improving efficiency is by restricting yet further the power of the language, for example by avoiding disjunction as well as negation. This is the major restriction imposed when formulating a specification in the algebraic style. In an algebraic specification, fresh names (such as "gcd") must be introduced for each function to be specified. The specification is formalized as a set of algebraic equations, each of which must be true of all values of the variables they contain. These equations are connected implicitly by AND. Only this form of conjunction is allowed—no negation and no disjunction. As a result, it is even more difficult or even impossible to write a program which preserves the clause structure or content of the original requirements. Instead, the algebraic equations have to be derived as needed by mathematical reasoning from the whole of the original specification. This is illustrated in the case of the greatest common divisor:

One way of improving efficiency is by restricting the power of the language.

L3.1 The greatest divisor of x is x . So the greatest common divisor of x and x is also x :

$$x = \text{gcd}(x, x) \quad \text{for all } x$$

L3.2 If z divides x and y , it also divides $x + y$. So every common divisor of x and y is also a common divisor of $x + y$ and y . Similarly, every common divisor of $x + y$ and y is also a common divisor of x and y . So the greatest of these identical sets of common divisors are the same:

$$\text{gcd}(x, y) = \text{gcd}(x + y, y) \quad \text{for all } x, y$$

L3.3 Every common divisor of x and y is also a common divisor of y and x :

$$\text{gcd}(x, y) = \text{gcd}(y, x) \quad \text{for all } x, y$$

The three laws given above can serve as an algebraic specification of the gcd function. The consistency of the specification has been guaranteed by proof, which shows that the laws follow from a set of requirements already known to be consistent. But the question remains, are the laws a complete specification, in the sense that there is only one function satisfying them? Or do we need to look for more laws? A proof of completeness has to show that for any given positive numerals p and q there is a numeral r such that the equation

$$r = \text{gcd}(p, q)$$

can be proved solely from the algebraic specification and the previously known laws of arithmetic.

This can be shown by mathematical induction: We assume the result for all p and q strictly less than N , and prove it for all p and q less than or equal to N . For such numbers, four cases can be distinguished:

(1) Both p and q are strictly less than N . In this case, what we have to prove is the same as the induction hypothesis, which may be assumed without proof.

(2) Both p and q are equal to N . Then the result

$$N = \text{gcd}(p, q)$$

is proved immediately by law L3.1.

(3) $p = N$ and $q < N$. It follows that $p - q$ is positive and less than N . By the induction hypothesis, there is an r such that

$$r = \text{gcd}(p - q, q)$$

is deducible from the algebraic laws. One application of L3.2 then gives

$$r = \text{gcd}((p - q) + q, q)$$

which by the laws of arithmetic leads to the required conclusion:

$$r = \text{gcd}(p, q)$$

(4) $p < N$ and $q = N$. Then there is an r such that

$$r = \text{gcd}(q, p)$$

is provable in the same way as in case (3) described above. One application of L3.3 then gives

$$r = \text{gcd}(p, q)$$

That concludes the proof that the algebraic specification is complete.

Clearly there is no structural correspondence between the three clauses of the algebraic specification and the five clauses expressing the original requirement. As a result, some mathematical ingenuity and labour has been needed to prove that the two orthogonal specifications describe (and completely describe) the same function. This labor could be avoided by simply leaving out the original formalization of requirements in the general notations of mathematics, and by starting instead within the more restricted equational framework of algebra.

But this would be a mistake. In the example we have been considering, the mistake can be explained as follows. The purpose of the specification is to tell the user of a subroutine the properties of the result it produces, and to do so in a manner conducive to the wider objectives of the program as a whole. Clearly, the user of a subroutine to compute the greatest common divisor will be very directly interested in the fact that the result of every subroutine call divides each of its two arguments exactly. But the algebraic law tells us only that the same result would have been obtained if the two arguments had been permuted (L3.3) or added together (L3.2) before the call. These facts by themselves seem a lot less directly useful.

It would also be a mistake to regard the different specifications, the abstract one and the algebraic one, as rivals or even as alternatives. They are both needed; they are essentially complementary, and they can be used for different purposes at

different stages in the progress of a software project. Algebraic laws are relevant in the design of an efficient algorithm; they are useful even at an earlier stage, because they help to decide how a program should deal with extreme or exceptional cases. For example, if one of the arguments of the *gcd* subroutine is zero or negative, the result should obey the same laws as the greatest common divisor of positive numbers, as far as mathematically possible.

For specifications of the highest quality and importance I would recommend complete formalization of requirements in two entirely different styles, together with a proof that they are consistent or even equivalent to each other. For example, a programming language can be defined axiomatically, by giving methods of proving correctness of programs expressed in the language. A language can also be defined algebraically, by giving algebraic laws from which equivalence of programs can be proved. A language for which two consistent and complementary definitions are provided may be confidently taken as a secure basis for software engineering.

Like the original requirements specification, a set of algebraic laws can be input to the British Museum computer, which will add these laws to the previously known axioms of mathematics and then start deriving theorems from them. Provided that the laws are complete (as we have proved our laws for *gcd* to be), the computer will eventually discover any fact that we want, for example, that the greatest common divisor of three and two is one. Furthermore, the amount of time we have to wait for this earth-shattering result will be millions of times less than the original British Museum algorithm applied to the original requirements. But we would still have to wait too long—on reasonable assumptions, the whole universe will reach a uniform temperature around four degrees Kelvin long before any interesting calculation is complete.

Functional program

Again an enormous increase in efficiency is required. And again, it can be obtained by restricting yet further the set of notations in which the mathematical equations are expressed. These restrictions are designed to prevent the pursuit of deductions irrelevant to the required result. That is achieved by use of an applicative or functional programming language.

The abstract specification and the algebraic one are not rivals or alternatives.

Here is a functional program to compute the greatest common divisor of positive integers:

F4.1 $\text{gcd}(x, y) = x$ if $x = y$

F4.2 $\text{gcd}(x, y) = \text{gcd}(x - y, y)$ if $x > y$

F4.3 $\text{gcd}(x, y) = \text{gcd}(y, x)$ if $x < y$

In structure and content, this is very similar to the algebraic laws, but it conforms to the restrictions imposed by a functional programming language. A typical restriction is that the left-hand side of each equation must consist of a single application of the operator being defined (*gcd*) to distinct simple parameters (x and y). There is no such restriction on the right-hand side, which may even contain occurrences of the operator being defined. In the evaluation of an expression, a computer treats each equation of the program as a substitution rule. At each application of the rule, a call of the function is replaced by a copy of the right-hand side of the definition, with appropriate substitution of arguments for parameters.

For example, here is a sequence of substitutions which calculates the greatest common divisor of 10 and 6:

$\text{gcd}(10, 6) = \text{gcd}(4, 6)$	by F4.2
$= \text{gcd}(6, 4)$	by F4.3
$= \text{gcd}(2, 4)$	by F4.2
$= \text{gcd}(4, 2)$	by F4.3
$= \text{gcd}(2, 2)$	by F4.2
$= 2$	by F4.1

This trace of execution of a functional program is nothing but a proof of the fact that $\text{gcd}(10, 6) = 2$. It is the same proof as would eventually be discovered by the British Museum algorithm. But the British Museum algorithm starts from the axioms and generates vast numbers of hopelessly irrelevant truths. The implementation of a functional programming language starts from the left-hand side of the theorem to be proved, and every intermediate step should lead directly towards the goal. However, the responsibility for controlling this goal-directed behavior is placed upon the programmer, who has to prove that

there is no infinite chain of substitutions. For example, as an algebraic formula,

$$\text{gcd}(x, y) = \text{gcd}(y, x)$$

is quite correct, but if this is executed as part of a functional program, it leads to an infinite chain of substitutions. In the program shown above, this cycle is broken by ensuring that the dangerous substitution is made only when y is strictly greater than x .

Proof of termination of a functional program is very similar to proof of completeness of algebraic equations. It requires knowledge of the domain of application, and it may also require knowledge of the strategy used by the language implementor for selecting between substitutions when more than one is possible. A simple strategy (of lazy evaluation), which is also the one that most often succeeds, is not generally the most efficient one.

Let us now consider how long the program will take to terminate. In the calculation of $\text{gcd}(N, 1)$, the number 1 will be subtracted from N until it reaches 1, and this will be done $N - 1$ times. On a typical 32-bit microprocessor, N is limited to about 10^{10} , so the calculation will take a few hours, which might be acceptable for a program or prototype intended to be used only once. On our largest and fastest supercomputers, numbers range up to about 10^{20} , and we might have to wait around a million years or so for an answer. It seems we are still left with a problem of efficiency, and we need to look for an improvement of at least twenty orders of magnitude.

Optimization

Use of a more restricted and more efficient programming language might gain us one or two orders of magnitude in efficiency, which is hardly worthwhile. What we need is a method of calculation inherently faster than the one we have first stumbled upon; for example, one requiring a number of steps proportional only to the logarithm of the arguments to which it is applied. For this, we need to go back to the earlier stage of analyzing the algebra, and take advantage of our knowledge of the nature of the mechanism used to execute the algorithm. On modern computers, division and multiplication by two are extremely fast, and it is equally fast to test whether a number is odd or even. We are therefore led to derive algebraic laws involving these operations.

Three new laws will be sufficient:

L5.1 If z divides x , then $2z$ divides $2x$. So if z is the greatest common divisor of x and y , then $2z$ is a common divisor of $2x$ and $2y$. It is therefore not greater than their greatest common divisor:

$$2gcd(x, y) \leq gcd(2x, 2y)$$

Conversely, if z is the greatest common divisor of $2x$ and $2y$, then z is even and $z/2$ is a common divisor of x and y :

$$gcd(2x, 2y)/2 \leq gcd(x, y)$$

From these two inequations it follows that

$$2gcd(x, y) = gcd(2x, 2y)$$

L5.2 All divisors of an odd number are odd, and if an odd number divides $2x$ it also divides x . If y is odd, the greatest common divisor of $2x$ and y is odd, so it is also a common divisor of x and y :

$$gcd(2x, y) \leq gcd(x, y) \quad \text{if } y \text{ is odd}$$

Conversely, every divisor of x divides $2x$:

$$gcd(x, y) \leq gcd(2x, y)$$

From these two inequations it follows that

$$gcd(2x, y) = gcd(x, y) \quad \text{if } y \text{ is odd}$$

L5.3 If both x and y are odd, and x is greater than y , it follows that $x - y$ is positive and even. So under these conditions,

$$gcd(x, y) = gcd((x - y)/2, y)$$

Similarly, if x and y are odd and y is larger,

$$gcd(x, y) = gcd((y - x)/2, x)$$

When these equations are coded as a functional program, it becomes clear that the number of operations required when the argument is of size 2^N has been reduced to about N . The question arises whether this is the very best that can be done, or whether it might be worth trying to find a better algorithm. This is the kind of question studied by complexity theory. But experts in complexity theory tell me that they do not know whether a better algorithm for gcd exists, so we shall have to do the best we can with this one.

Procedural program

Pursuit of the very highest efficiency yet again requires adoption of a restricted notation in which conjunction is disallowed. Such a restriction is imposed by a procedural programming language like Fortran or Pascal. The component parts (P and Q) of a procedural program are joined not by any propositional connec-

Sequential composition is the secret of the efficiency of procedural programs.

tive, but rather by sequential composition, usually denoted by a semicolon:

$P; Q$

which instructs a computer to execute P first, and to continue with Q only when P terminates.

Sequential composition is the secret of the efficiency of procedural programs. All of the computing resources, storage, and communication capacity which have been used by the earlier component P become available immediately, without overhead of reclamation and reallocation, for reuse by the later component Q . However, the responsibility for planning the use and reuse of resources is placed on the programmer, and much opportunity is offered for subtle errors. Perhaps the main attraction of functional and logic programming is that the claiming of new resources and the reclaiming of unused resources are automatic, once the overhead of garbage collection is accepted.

The replacement of conjunction by composition means that the sequential structure of a program will in general be radically different from the conjunctive structure of a specification, and the necessary correspondence between them must be established by mathematical calculation or proof. Fortunately, these calculations can be carried out stepwise during the early stages of the design of a program, and each design decision can be proved correct before embarking on the next. Each design step transforms the structure of the specification to correspond more closely to the structure of the eventual program. After this, work on the component parts of the structure can proceed concurrently and independently, because each of them has been fully specified, with the same precision as the whole of the original requirements. When the implementations of the parts are delivered, they can be assembled together with reasonable confidence that the system as a whole will work. This confidence is obtained not by laborious inte-

gration testing when the code is delivered, but rather by a proof that was conducted before a word of code was written. Reliable assembly of prespecified parts is an essential mark of maturity in any engineering discipline.

A procedural program or subprogram can be specified by a pair of assertions, a precondition and a postcondition. A precondition describes properties of the program variables which may be assumed to hold when the program starts. For example, let the lowercase letters x and y stand for the values of the arguments supplied on entry to the gcd subroutine. The precondition states that these are positive:

$$P6.1 \quad x > 0 \wedge y > 0$$

In fact, the program will not change the values of x and y , so this assertion remains true throughout execution.

A postcondition describes properties of the program variables that must be true when the program terminates. For example, let Z be the program variable introduced to hold the result of the subroutine. Then the postcondition is

$$P6.2 \quad Z = gcd(x, y)$$

The first step in the design of a sequential program ($P; Q$) is to formalize separate specifications of P and of Q and to prove that the combination of these separate specifications will meet the original specification of the whole program. This is easily achieved if

(1) the precondition of P is the same as the precondition of the whole program (or is implied by it);

(2) the postcondition of Q is the same as the postcondition of the whole program (or implies it); and

(3) the postcondition of P is the same as the precondition of Q (or implies it).

So the design of a sequential program involves only the discovery of an appropriate intermediate assertion which will be true when control passes the semicolon which separates the parts of the program. The intermediate assertion may contain new program variables local to the subroutine, which are introduced to assist in the calculation; we will denote these by capital letters, for example, X , Y , and N . In general, the discovery of a suitable intermediate assertion requires all the skill and judgment of the program designer. Though there are a number of useful heuristics, this is not the right place to explain them; so without further ado, here is an intermediate assertion for the example program to compute the greatest common divisor:

$$P6.3 \quad 2^N \gcd(X, Y) = \gcd(x, y)$$

Now the two parts of the program may be considered independently. The task of the first part is to make P6.3 true on termination. That is easily accomplished by just one multiple assignment:

$$N, X, Y := 0, x, y$$

If proof is needed, it may be obtained by substituting the final values for the assigned variables in the postcondition, and observing that

$$(2^0) \gcd(x, y) = \gcd(x, y)$$

The second task is harder, and will again be split into two parts by the intermediate assertion

$$P6.4 \quad 2^N Z = \gcd(x, y)$$

The second of the new subtasks would already be accomplished if N were zero. If N is non-zero, it can be made closer to zero by subtracting one. But that would make P6.4 false, and therefore useless. Fortunately, the truth of P6.4 can easily be restored if every subtraction of one from N is accompanied by a doubling of Z . This can be checked if it is felt necessary by proving that

$$\begin{aligned} n > 0 \wedge 2^N Z = \gcd(x, y) \\ \Rightarrow 2^{N-1} (2Z) = \gcd(x, y) \end{aligned}$$

Since termination is obvious, we have proved the correctness of the loop

while $N > 0$ **do** $N, Z := N - 1, 2Z$

On termination of this loop, the value of N is zero, and P6.4 is still true. Consequently, the postcondition of the whole program has been established.

Having completed the first and last of the three tasks, the time has come to confess that the middle task is the most difficult. Its precondition is P6.3 and its postcondition is P6.4. I suggest that the task be split yet again into four subtasks, in accordance with the following series of intermediate assertions:

$$P6.3 \wedge (X \text{ odd} \vee Y \text{ odd})$$

$$P6.3 \wedge (Y \text{ odd})$$

$$P6.3 \wedge (Y \text{ odd}) \wedge X = Y$$

The coding of these parts can be derived almost entirely from the algebraic laws, and is left as an exercise.

When this coding has been done, we will have a complete program expressed in some mathematically meaningful programming notation such as Pascal. In suitable circumstances, a reliable automatic translator will be available to translate the program into binary machine code of an available computer, and this will be fast enough for most general purposes. If still higher speed is required, it may be

The need for rapid implementation of prototypes requires the talents of an experienced hacker rather than the formal precision of an engineer.

necessary to carry out yet another stage of formal design, optimization and verification. For example, the Pascal program may be converted into the even lower level language of a horizontal microcode, or other hardware description for custom-built VLSI. In this context, the cost of design errors is extremely high, and the competent hardware engineer will warmly welcome the extra confidence that can be gained by the formal manipulations and proofs conducted at earlier stages in the design.

In a software project, the process of formal design may also have a beneficial influence on delivery dates and on reliability of the finished product. This will be important in safety-critical applications. But as in other branches of engineering, the full benefit of documentation will be most clearly felt in the long years of maintenance following first delivery of a large program. Apart from easing the removal of any remaining errors, a well documented program can be more readily and confidently improved or altered to meet changing requirements. As in other branches of engineering, the documentation should be kept up-to-date. This is likely to become more and more difficult as the changes accumulate, and when it becomes impossible, this should be the signal that the time has come to rewrite or replace the whole program.

In selection of formal notations for the various phases of a software engineering project, the following criteria are relevant:

- (1) For original capture of requirements, only clarity and precision are of importance.
- (2) In intermediate documentation used by the implementor, we have the related

objectives of clarity and correctness.

(3) In finally delivered code, we require correctness and also seek efficiency of execution.

The final code of the program must, of course, be expressed in a formal notation that can be automatically translated and executed by computer. But there is no need for any of the preceding documents to be executed, or even input to a computer. Their main purpose is clarity and convenience for communication, calculation, and proof. A premature insistence on execution (on a less than cosmological time-scale) may stand in the way of these more important objectives.

It is obviously sensible to use for final coding a language which is as close as possible to that of the original specification, so that the number of steps in the design process is kept small. But the programming language must not be so inefficient that clever and complicated coding tricks are needed to compensate. This seems to be a major problem with logic programming. My view is that a modern functional programming language can provide a nice compromise between the abstract logic of a requirements specification and the detailed resource management provided by procedural programming.

Such a high-level language may also be used for rapid implementation (partial or total) of the original requirements, and so provide a rapid check of the adequacy and completeness of the specification of requirements. A similar role is played by wooden models and perspective drawings in architecture or car design. But it is advisable to plan to throw away such models after use. The overriding need for rapid implementation gives scope for the talents of an experienced hacker rather than the formal precision of an engineer, and it is unlikely that the model can be taken as the basis or framework for subsequent development of the delivered product.

The example which I have used to illustrate these conclusions is one which has been used over and over again in textbooks on computing science. Because it is such a small and familiar example, it does not complicate the explanation of more general and more important ideas. For this reason, it has also revealed (all too clearly) the full weight of the notations and complexity of the mathematical proofs involved in formalization of the process of program design. The reader may well be discouraged from applying these methods to problems of a scale more typical of software engineering. And there are many

other serious concerns which are not addressed directly by formalization, for example, cost estimation, project management, quality control, testing, maintenance, and enhancement of the program after delivery.

Nevertheless, I conjecture that such pessimism would be premature, and that a more formal understanding of the decisions involved in program design will provide a more secure framework within which solutions to the other problems can be fitted. The reason for my optimism is that formal methods find their most effective application in splitting large and complex projects into shorter phases, and in splitting large and complex products into smaller components, which may then be designed and implemented independently. This splitting may be repeated as often as necessary, until each subtask is no larger and no more complex than the simple example treated above.

And maybe this was not such a simple example after all. The reader who finds it too simple is invited to improve the program so that it accepts and deals sensibly with negative numbers and zero. There are two ways of doing this:

(1) The traditional method of program maintenance is to re-test the final program on the new range of arguments, change the program as little as possible, and describe the resulting program behaviour in the user manual.

(2) The method which I recommend is to reexamine the specifications and proofs presented above, to see where they have relied on the assumption that the arguments are positive; these are the places where the design and the program must be altered. This method is in accordance with sound engineering principles, and is therefore more likely to deliver a product of high quality.

The main purpose of this article has been to show the all-pervasive role of mathematical reasoning throughout all stages of a product life cycle, and to show the need for an appropriate range of mathematical notations to communicate design decisions with other people, with a computer, and even with oneself. □

Acknowledgments

To Richard Karp for advice on computational complexity, and to all seven referees of this article.

Bibliography

This brief overview of formal methods for program design has necessarily omitted some important topics, for example, data refinement. No mention has been made of concurrent or distributed programming, for which some kind of formal development method is almost essential. No adequate instruction has been provided to enable the reader to put into practice the methods described. And finally, no acknowledgment has been made to the brilliant researchers who have contributed to progress in the field. I hope the reader may remedy some of these deficiencies by consulting some of the books listed in this personally selected bibliography.

1. Requirements

Specification Case Studies, Ian Hayes, ed., Prentice-Hall International, U.K., 1987.

2. Logic programming

R. Kowalski, *Logic for Problem Solving*, North Holland, Amsterdam, 1979.

K.L. Clark and F.G. McCabe, *micro Prolog: Programming in Logic*, Prentice-Hall International, U.K., 1984.

3. Algebra

J.J. Martin, *Data Types and Data Structures*, Prentice-Hall International, U.K., 1986.

4. Functional programming

P. Henderson, *Functional Programming: Application and Implementation*, Prentice-Hall International, U.K., 1980.

H. Abelson and G.J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Mass., 1985.

5. Procedural programming

D. Gries, *The Science of Programming*, Springer-Verlag, New York, 1981.

R.C. Backhouse, *Program Construction and Verification*, Prentice-Hall International, U.K., 1987.

6. Data refinement

C.B. Jones, *Systematic Software Development Using VDM*, Prentice-Hall International, U.K., 1986.

7. Parallel programming

C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, U.K., 1985.

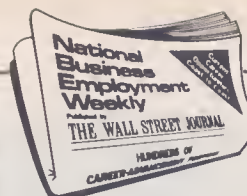
K.M. Chandy and J. Misra, *Parallel Program Design: a Foundation*, Addison-Wesley, Reading, Mass., forthcoming.

8. Computational complexity

A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

9. General

Programming Methodology, D. Gries, ed., Springer-Verlag, New York, 1978.



Jobs for computer pros!

Every week, the National Business Employment Weekly, published by The Wall Street Journal, contains hundreds of high-paying jobs from all across the country, including career opportunities in virtually every area of computer technology.

PLUS...weekly editorial features covering every aspect of career advancement.

"Computer" Issue.

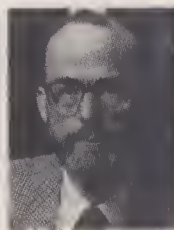
October 4th.

Extra career opportunities for computer professionals.

Pick up the National Business Employment Weekly at your newsstand today. Or we'll send you the next eight issues by first class mail. Just send a check for \$35 to:

National Business Employment Weekly
Dept. IC2, 420 Lexington Avenue
New York, NY 10170

Reader Service Number 2



Charles Antony Richard Hoare began his career in 1960 writing computer programs for Elliot Brothers, Ltd., in England. During his eight years there, he designed everything from simple subroutines to computers and high-level programming languages. He left the industry in 1968 to teach computer science at the Queen's University in Belfast, Ireland, moving to Oxford's Programming Research Group in 1977, where he is now Professor of Computation.

Hoare received the Turing Award in 1980 and is noted for his contribution to the formal definition of programming languages by means of axiomatic semantics. He has made the establishment of computer programming as a serious professional discipline a driving force in his work. He points to his development of the concept of the communicating sequential process as one of his major achievements; it brings together high-level mathematical theories with the design of languages and architectures for multiprocessor networks.

Readers may write to Hoare at the Dept. of Computer Science, University of Texas at Austin, Taylor Hall 2.124, Austin, TX 78712-1188. After September, write to C.A.R. Hoare, Oxford University Computing Laboratory, 11 Keble Rd., Oxford, OX1 3QK, UK.

IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS

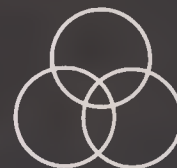
Sponsored by: **The Computer Society of the IEEE and IEEE Circuits and Systems Society**
In cooperation with: **The IEEE Electron Device Society**

ICCD '87

Rye Town Hilton, Rye Brook, New York
October 5-October 8, 1987

THE INTERDISCIPLINARY CONFERENCE

- ☐ **VLSI and Technology**
- ☐ **Design and Test**
- ☐ **Computer-Aided Design**
- ☐ **Architectures and Algorithms**



The focus of this multi-discipline conference is intended to emphasize the interactions between system design, logic circuit design, memory design, architecture, software, CAD, Testing, Physical Design, and VLSI Technology. The objectives of this conference will offer to those attending an alternative view into the VLSI CAD environment. Unlike other conferences focusing on the generalized aspects of DA, ICCD stresses the interrelationships which exist in the CAD industry.

HIGHLIGHTS

41 SESSIONS ■ Algorithms ■ Object-Oriented Techniques for CAD ■ Synthesis and Floorplanning for VLSI ■ Methods in Test Pattern Generation ■ Multiprocessors ■ Placement Techniques on Parallel Architectures ■ Local Area Network Controllers ■ System Packaging ■ Computer Design ■ Synthesis of DSP Systems at Leuven ■ DEC 32-Bit Processor Family ■ National Semiconductors NS32532 Microprocessor ■ Symbolic Layout Compaction Benchmarks ■ Self-Timed Design ■ VLSI Technologies ■ Special-Purpose Architectures ■ Routing ■ Knowledge-Based Approaches for Design and Test ■ Reliable Systems ■ Graphics and Image Processing ■ New Approaches in VLSI Design ■ Storage Technology for the 90's ■ Digital Signal Processing ■ Integrated CAD Systems ■ Design of Arithmetic Components ■ Physical Verification ■ Built-In-Self-Test in Practice ■ State-of-the-Art Technology Integration ■ Fault-Tolerant Computing ■ Symbolics Ivory Lisp Processor ■ Performance-Oriented Design ■ CCI Regulus Supermini ■ High-Level Synthesis ■ Design of MC8030 Microprocessor ■ High-Level Language Architectures ■ Advances in Physical Design ■ New Methods in Design ■ Architecture of Arithmetic Units ■ Logic Simulation ■ PLA Design and Testing

PLENARY SESSIONS

- **Design & Test**
Semiconductor and Silicon Compilers effect on Design and Test: Trends and Directions
- **CAD**
Computer Aided Fabrication of Integrated Circuits
- **VLSI & Technology**
The Development of High TC Superconductors and Their Application Potential
- **Architectures**
TBA

ICCD '87

AIRLINE INFORMATION

An arrangement with the Computer Society of the IEEE's International Conference on Computer Design and United Airlines offers you the services of its toll-free reservations desk along with the compliments of discounts.

5% on any fare for which you qualify including First Class or maxi-saver. The discount can range from 40% to 70% off normal coach fares.

Those not qualifying for the above discount, may be qualified for a minimum of 40% off the coach fare with no minimum stay or advance purchase required.

To make a reservation for one of the above or for any promotional fare, simply follow these easy steps:

- 1) Call United's toll free number: 800-521-4041 7 days a week, 8am-11pm EST.
- 2) Give the IEEE-CS Account #: 7146C
- 3) United will arrange to mail tickets to your home or office, or you may purchase them from your local Travel Agent.*

*If you purchase from your local Travel Agent be sure you or your travel agent calls the United's Convention desk to make your arrangement. The special IEEE-CS fare is available only for the ICCD '87 Conference thru United's Convention desk.

NOTE: Mileage plus will receive full credit.

HOTEL RESERVATION FORM

IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN - ICCD '87

Name _____
Street _____
City/State/Zip _____
☐ Single room
☐ Double room - Additional occupant: _____
☐ Charges on one bill
☐ Bill separately

Arrival Date & Time _____ Departure Date _____

Credit Card Name _____

Credit Card No. _____ Expiration Date _____

Cardholder's Name _____

Signature _____

Return To: Front Office Manager, ICCD '87
The Rye Town Hilton
699 Westchester Avenue
Rye Brook, New York 10573
Phone: (914) 939-6300

To guarantee your reservation, either a check for one night's lodging, or appropriate credit card information, must be included with the hotel reservation form.

Notes: Please indicate accommodations desired. Rooms should be claimed by 6 pm. We will attempt to accommodate you first at the Rye Town Hilton (914-939-6300). (Single \$102/Double \$120). It is your responsibility to find your own roommate. Reservations must be received no later than September 14, 1987. After that date reservations must be made directly with hotels on an availability basis. If reservations are made by telephone you must identify ICCD '87.

REGISTRATION FORM ICCD '87

	IEEE MEMBER	NON-MEMBER	STUDENT	AMOUNT
REGISTRATION				
Until September 18	\$160	\$200	\$65 (w/o Proc.)	_____
After September 18	\$210	\$265	\$65 (w/o Proc.)	_____
BANQUET	\$32	\$32	\$32	_____
LUNCHEON	\$14 per day	\$14 per day	\$14 per day	_____
☐ October 5, 1987	☐ October 6, 1987	☐ October 7, 1987		
			TOTAL	_____

IEEE or IEEE Computer Society Membership No. _____

Name _____ Organization _____

Street Address _____

City/State/Zip _____

Credit Card Name _____ Credit Card No. _____ Expiration Date _____

Cardholder's Name _____ Signature _____

Return To: Ms. Glenda D. McBride, Registrar, ICCD '87
Computer Society of the IEEE
1730 Massachusetts Ave., N.W.
Washington, D.C. 20036-1903
Phone: (202) 371-1013

To guarantee your reservation, either a check or appropriate credit card information (Visa, MasterCard, or American Express), must be included with the conference registration form.

Notes: Conference registration fee includes admission to the technical sessions, one copy of the conference proceedings (except students), break refreshments, and reception. Luncheons and banquet spaces are guaranteed only for advanced registrants. Otherwise, meals, literature, and other activities may be purchased at the conference on an as available basis. Refunds (less \$15 handling fee) will be made if request is received in writing by October 2, 1987. After that date no refunds can be made.

STANDARDS

Editor: Helen M. Wood, National Bureau of Standards, B154 Technology, Gaithersburg, MD 20899; (301) 975-3240.

IEEE Trial-Use Standards



Point/Counterpoint

An intense discussion of the use/misuse, purpose and value of IEEE Trial-Use Standards occurred during the March meetings of the IEEE CS Standards Activities Board and Standards Coordinating Committee. No conclusions were reached during either of those meetings. However, two of the outspoken individuals from those meetings have been invited to present their arguments here and to comment on the opposing viewpoint. The objective of

this discussion is to publicize these opposing views and to ask that you, the reader, provide us with your opinions on the subject.

The subject of this debate—the IEEE Trial-Use Standard—is described in the following paragraphs of the IEEE Standards Manual:

Where a draft has been generated that generally satisfies the standards developing group (i.e. subcommittee or working group) but needs input from a very broad



For

*Herbert Hecht
SoHaR, Inc.*

The importance of voluntary standards (such as IEEE Standards) to our society, individual industries, and companies makes it necessary to establish safeguards against arbitrariness and bias in all procedures leading to the adoption of a standard. An important provision in the IEEE standards procedures is the ballot that the sponsor uses to request approval of the standard by a group with a balanced membership representing producer, user, and general interests. This ballot can be marked to indicate approval, disapproval, or abstention. Comments regarding desired changes can be made by those approving, and are required from all those who disapprove. A standard can be forwarded for consideration by the IEEE Standards Board only if at least 75 percent of the balloted body voted, and if at least 75 percent of those who voted (other than to abstain) approve the standard. These provisions apply to both full-use and trial-use standards; they are intended to ensure that the standard represents a consensus of the affected users and producers of the product or services covered by the document.

In addition, the sponsor is required to attempt to resolve all negative ballots and comments that were received during balloting. This is an important safeguard against the adoption of a flawed or unfair standard based on votes of an uninformed or disinterested majority. However, this safeguard also permits

interests who might be put at a disadvantage by the adoption of a standard, whatever its technical merits, to delay its consideration by the Standards Board. Because reasonable time has to be allowed for responses, and because the standards process is supported by volunteer efforts, the resolution of objections, especially with not completely cooperative balloters, can drag on for a year or more. In the fast moving computer field, such a delay can cause great harm to the majority that wants to implement a standard.

The difference in the adoption process between trial-use and full-use standards lies primarily in the degree to which negative comments have to be resolved. For full-use documents, the Standards Board generally requires that objections be either resolved, or shown to be frivolous or motivated by considerations outside the technical area. The Standards Board is much more sensitive to harm that can arise from the adoption of a flawed standard than from delay in the adoption of a meritorious one; therefore, the burden of proof relative to unresolved objections is very much upon the sponsor. On the other hand, for trial-use standards, the requirement for resolution of objections is less stringent. The IEEE Standards Manual (par. 3.7) provides that "Proposed standards receiving a significant number of unresolved negative votes should be considered by the sponsor for Trial-Use."

What then are the limitations that arise from the adoption of a trial-use standard? First of all, there is the designation itself, which might dampen the enthusiasm of producers to provide

equipment or services in accordance with the standard, and which might also cause users of the equipment or services covered by the standard not to require compliance. Then there is a time limitation of two years on the validity of the standard. During this period the original objections, as well as any adverse comments received during the trial-use period, must be resolved to the satisfaction of the Board. If this resolution cannot be achieved, the standards effort must be re-examined. On the other hand, the exposure that the standard receives during the trial-use period can serve as an effective criterion for establishing its merit and significance to the industry.

Within the Computer Society a number of standards have initially been issued on a trial-use basis and later adopted for full use. In other instances, documents were kept in the draft stage for many months or even years to resolve objections which were not without technical merit but had comparatively little impact on the overall implementation or acceptance of the standard. Eventually, these objections were resolved, but with much more delay and dissension than would have been necessary if the trial-use route had been explored.

After balloting has produced the required majority vote in favor of a standard, the sponsor is faced with three possible situations:

- Objections are few and appear capable of resolution: This situation should lead to a decision to proceed with processing for a full-use standard.

(continued on page 96)

constituency, such a draft may be processed as an IEEE Trial-Use Standard. This is a preferred alternative to the widespread distribution of unapproved drafts. For approval, such a draft requires a letter ballot of the Sponsor and approval by the Standards Board as a trial-use standard.

When a Sponsor is unable to resolve negative ballots to a satisfactory level, or uncertain aspects of the document justify preliminary distribution, it may consider submission of the draft to the Standards Board as a trial-use standard.

When the Standards Board cannot attain a suitable level of approval for a draft submitted for adoption as an IEEE Standard, it may decide to approve it as a trial-use standard.

Presenting the argument for the use of trial-use standards is Herb Hecht, president of SoHaR, Inc., past chair of the Standards Committee of the IEEE CS, and active participant for over 10 years in the development of standards within the IEEE CS. Opposing the use

of trial-use standards is Gary Robinson, Senior Manager of Corporate Standards for Digital Equipment Corporation, Vice-Chair of ISO TC97 (Systems and Application Elements), member of the IEEE Standards Board, and long-time participant in LAN standards development.

Laurel V. Kaleda
Chair, Standards Coordinating
Committee



Against

Gary Robinson
Digital Equipment
Corporation

The concept of a trial-use standard is somewhat similar to the concept of a trial-use airline—there is no indication that either one will fly, but both are busy booking passengers. Occasionally, they will work—usually, the passengers are left standing, trying to remember why they thought saving a few dollars would be worth the hassle of dealing with the trial-use concept. This article looks at trial-use standards from a slightly prejudiced viewpoint—I do not believe that they, like the trial-use airline, are worth the problems for the possible gain that can be achieved.

Before I comment upon trial-use standards, however, it will be helpful to look at the normal consensus standards process. This is definitely a lengthy process, with a series of checks and balances to insure the integrity of the idea, as well as the validity of the solution, within the Information Technology (IT) market. The reason for these checks and balances is to insure that the standard is well thought out when it hits the public, to provide the market time to become accustomed to the concept of a possible standard, and to insure that there is no singular overwhelming force that drives the standards creation.

A simple taxonomy of standards would indicate two types of standards—focused standards and systems standards. Focused standards, such as character code standards, tend to be defined and implemented, and are by definition inherently correct. There is

little ambiguity or misinterpretation possible.

Systems standards, however, must be validated before they can be termed “standards.” There is no unambiguous solution, and the ambiguity must be dealt with before the standard is issued. It is in the arena of systems standards that the full standards process becomes valuable. The review, although painful at times (and painfully slow at others), does provide that all parties who have a material and direct interest can critique the standard before it is published. Groups use working documents—labeled as working documents—to initiate testing and validation, knowing that there is a high probability that change will occur. The documents are kept within the community working on them and they look very informal. While there is a temptation to design products to a working document, there is the check and balance process that indicates that this is done at the designer’s peril. Key to the standards process is involvement at this stage, and because the process is open to all, the comments reflect the thinking of the IT community, or at least those members of the community concerned enough to become involved. If there is no interest, the standard simply disappears; otherwise, the standard, with complete documentation and community acceptance, is published. If, after publication, the document is found to contain errors, ambiguities, or other problems associated with the imperfect real world, there are revision procedures as well as other procedures in place to insure that these problems are dealt with.

The IEEE trial-use standard, on the

other hand, masquerades as a “published” standard. The trial-use standard is ostensibly published to encourage testing and implementation prior to the completion of the check and balance system. It is published in a format similar to that of the IEEE standards and is bound. Once this semistandard is implemented, it becomes difficult for any person, no matter how talented, to recall and to fix. This process is akin to the jurors in a trial being told to disregard previous testimony—the testimony is never quite forgotten or disregarded. Although the printed volume carries a caveat in the front that it is a trial-use standard, subject to revision, the caveat is much too easily ignored. As someone once said—“If you quack, waddle, and have webbed feet, people are gonna think you’re a duck even if you have a sign around your neck that says ‘I’m a goose.’”

I believe that a trial-use standard can be abused to circumvent the normal checks and balances of the consensus standards procedure. An official IEEE printed document is assumed to be a standard, no matter what caveats are printed upon the cover. There appears to be an assumption that if enough people seize and implement a trial-use standard, it will become a “de facto” standard which will then build enough pressure to create a “de jure” standard. This is the Peter Pan theory of standards—“If you wish real hard, the standard will come true.” After all, it worked for Tinker Bell.

Unfortunately, as the standards manager of a major corporation, I cannot

(continued on page 96)



For

(continued from page 94)

- There are a number of objections, and it is not certain how much time and effort will be required to resolve these: This is a situation where a trial-use standard should be considered as described below.

- There are numerous or very substantive objections: In this case, redrafting the standard and possibly restricting its scope are desirable.

For the second condition that is listed above, the immediate action must be an attempt to respond to all comments, particularly those that accompany the negative votes. The IEEE Standards Manual requires this procedure, and the Standards Board will not consider the project (for either trial or full use) until it is satisfied that a sincere effort has been made to reach an agreement with the dissenters. When resolution cannot be reached for a significant number of negative ballots, the respondents whose comments could not be satisfied may be informed that approval of the standard for trial-use is being sought. In some cases, this step may help negotiations so that resolution can be achieved and the full-use processing can resume. In other cases, an application for trial-use will be considered by the Board which will be furnished all records (ballots, comments, correspondence, etc.) that pertain to the project. The Standards Board may also independently contact (or be contacted by) the originators of the unresolved comments to obtain direct insight into the basis for the difficulties in reaching an agreement.

Thus, the trial-use standard is a useful mechanism in establishing consensus, in bringing emerging standards into the public domain, and in ensuring fairness to all participants in the standards process. It serves a very useful purpose and should be retained.

Rebuttal

I read Herbert Hecht's article with interest. I feel that he understands the trial-use process very well and has the ability to explain the process with a high degree of lucidity. However, I believe that he has forgotten that standards are not an end in themselves, and that a standard serves only as a vehicle for insuring a fundamental commonality among all impacted groups. The scope of this response is not global, but limited to the use of trial-use standards in the Computer Society.

The root of the problem lies in the use of the word "unresolved." Hecht has interpreted this word to mean that a standard cannot be issued until all the "negative" votes are turned into "yes" votes. This, simply stated, is not the case. The correct concept—not semantics—is that all negative votes must be responded to, and the response must take the form of a professional response. If there is a technical difference, the basis of the difference must be recognized and understood, and a response written. There is no demand that the vote be changed—only that the need that drove the comment be understood. If the committee believes that the comment is valid, it should take steps to implement that change—professionalism requires no less. If the committee believes that the comment is not valid, it owes the originator a response on why it feels the way it does.

Trial-use standards do not do this. The trial-use process allows a standard with negatives which may have a significant impact to be published under the guise of a consensus standard. (The use of the term may be deliberate; a trial-use standard is unsure of the nature and impact of future changes.) If the concept being standardized is valid enough for publication, and if the process has been followed correctly, then the standard is good enough to be called a full-use standard, even with negative votes.

If the concept and process are not rigorous enough, then the standard is not good enough to be published and should go back to committee. The trial-use standard is equivalent to the third bit in a binary system—a "maybe bit." It is neither negative or positive.

The ultimate purpose of the standards process is to get the standard accepted, used, and implemented. The process is not an exercise in getting everyone to agree. Honest differences with the groups of producers, users, and general interests, and with the government will exist—differences that should be recognized and dealt with. A standard that is neither usable nor practical should not be considered for standardization. In response to Mr. Hecht's plea to retain trial-use, I would advocate the concept of the draft standard and distribute it in an unbound form. This procedure would encourage more use, more discussion, and more participation in the process. And it is this—participation by everyone—that makes the process viable.

Standards are practical solutions that should be implemented for a business reason, not academic solutions which are monuments to unusable perfection. As for the trial-use standard—it should be placed in the same category as the "maybe bit."

—G.R.

The ballot box

Let us know your position on IEEE Trial-Use Standards. Simply turn to the back of the magazine, circle the appropriate number on the Reader Service Card, and send the card in. We'll tally results and report the vote in the November issue.

RS #185 Herb Hecht has it right. I agree with most of his points, and I'm for Trial-Use Standards.

RS #186 Gary Robinson exposed the fallacies behind Trial-Use Standards. I'm against them, too.

RS #187 Neither argument was convincing. I'm still neutral or undecided.

**Use the Reader Service Card at the back of the magazine.
Now processed more frequently to ensure prompt replies.**



Against

(continued from page 95)

take the chance that Tinker Bell will once again fly. I cannot permit the design of a multimillion dollar architecture on a wing and a prayer. I need a standard—one that is valid, checked, and accepted. I believe that if something is important enough to enter the standards process, then it should have enough importance—and permanence—to complete the entire process. The trial-use concept can be seen as an attempt to circumvent the process, due to a perceived inefficiency in the process. The IEEE has not helped the problem; it has merely introduced another layer of confusion into the process. The question "When is a standard not a standard?" becomes important because of its potential impact on the entire standards process. If a concept is developed enough to be printed as a standard, then it should be good enough to

be a standard. If it can't stand on its own without qualifications and caveats, then it should not masquerade as a standard in IEEE standard's clothing. Only the IEEE, of all the standards development organizations in the world, has chosen to use the trial-use concept.

In conclusion, if I believed that all design engineers were familiar with the standards process, if everyone did not have critical schedules and deadlines to meet, if everyone understood that "trial-use" means "looks good right now, but maybe it won't really work," I would not have a problem with the concept. But we work in a real world, where there are more problems than real solutions. What the trial-use standard promises is another semi-real solution—an airline that might fly, might make schedules, might get you where you want to go. All of these might, however, make me use a real airline, where I know what my destination is, and how to get there. I am willing to make a commitment to the consensus process and help to improve it; I challenge the IEEE to "stan-

dardize" their approach to standards with the rest of the IT community and do the same.

Rebuttal

The limitations of trial-use standards have been acknowledged in the Pro presentation. The comparison of trial-use standards with a trial-use airline is not appropriate. The IEEE Standards Office can not recall an instance in which a trial-use standard has not progressed to a full-use standard. Trial-use standards originating within the Computer Society have undergone only minor changes in their transition to full-use standards. Past performance does not provide a guarantee that a contrary event might not occur in the future, but it indicates that the odds are a good deal more favorable than those confronting the purchaser of a ticket on a start-up airline.

The trial-use procedure is particularly important for the IEEE Computer Society because the society originates standards. ANSI and ISO adopt standards which were originated by other organizations (such as the IEEE). That the trial-use procedure is absent from the procedures of these organizations is not relevant to its use within our environment.

The greatest weakness in the Con argument is the absence of better alternatives. Widely circulated drafts are a much less rigorous and effective means for collecting comments, and they hardly provide greater incentive for a producer to invest the effort required for their implementation. Procedures for permitting the wide circulation of drafts (including their sale by the IEEE Computer Society) are much more controversial than those surrounding dissemination of trial-use documents. Keeping the draft under wraps until all negatives are resolved deprives both producers and users of the benefits which the proposed standard offers.

—H.H.

DIRECTOR OF ACADEMIC COMPUTING SERVICES UNIVERSITY OF CALIFORNIA, LOS ANGELES

UCLA invites applications and nominations for the position of Director, Academic Computing Services. The Director reports to the Administrative Vice Chancellor and is responsible for planning and managing all centrally provided academic computing services, including managing the operations of an IBM Model 3090 with two vectors, a Microcomputer Support Office, and a Network Development Group. Supervise facilities, personnel and operations. Position requires demonstrated working knowledge of an extensively distributed and diversified academic computing facility. Demonstrated managerial skills. This position is subject to the University Conflict of Interest Code required by the Political Reform Act of 1974. The candidate selected will be required to complete financial disclosure statements for public record.

Salary commensurate with experience and qualifications. UCLA offers an excellent benefits package that includes a variety of tax-deferred annuity programs. Please send resume, with cover letter indicating qualifications for the position, by September 18, 1987 to: Albert A. Barber, Vice Chancellor, University of California, Los Angeles, 3134 Murphy Hall (Job #R-3106), 405 Hilgard Avenue, Los Angeles, CA 90024-1405. Affirmative Action Employer.

UCLA

TOOLS FOR SOFTWARE QUALITY

From the leaders in modern software engineering technology

Eiffel™: The best object-oriented environment

- Multiple inheritance
- Generic classes, static typing, assertions, and much more
- Implementation through C for full portability
- An open environment, allowing reuse of existing software
- Support for the automatic generation of C packages
- Automatic incremental recompilation
- Tools for documentation, debugging
- The ideal system for developers of high-quality software
- Particularly appropriate for software tool developers

Cépage™: The multi-language structured editor

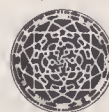
- Support for any language you want to use
- A highly visual interactive interface
- Input: menu-driven or just by typing (on-the-fly parsing)
- Good for programs, specifications, technical documents...
- Automatic support for your style standards and conventions
- Adapt to any language in a matter of hours

OUR SEMINARS: Object-oriented Design

- Paris (France), Sept 21-22 (in English), 23-24 (in French)
- Los Angeles (California), Oct. 21-22
- San Francisco (California) Nov. 19-20

Call us today at (805) 685-1006

270 Storke Road, Suite 7 - Goleta, CA 93117 (USA)
In Europe: Société des Outils du Logiciel, 3 bis cité d'Hauteville
75010 Paris (France) Téléphone 33/1-47 70 05 46
Cépage and Eiffel are trademarks of Interactive Software Engineering, Inc.



Interactive

Software Engineering, Inc.

Professional tools for the development of high-quality software

Reader Service Number 6

UPDATE

Contributions to Update are welcome. Send news of industrial or university research and of public policy or professional issues to *Computer Update* Editor, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 or to Bruce D. Shriver, *Computer* Editor-in-Chief, IBM T.J. Watson Research Center, PO Box H0-B04A, Yorktown Heights, NY 10598.

National Supercomputer Center assured funding until 1990; capacity doubled

Sporting a new unit that has doubled its capacity, the John von Neumann National Supercomputer Center has been assured of \$81.2 million in funding to continue operations until 1990.

The Princeton, New Jersey, facility run by the Consortium for Scientific Computing received a favorable review from the National Science Foundation this spring. As a result, the National Science Board has decided to fund the center for the length of the cooperative agreement that established it. The agreement, which runs until 1990, is for \$69.2 million.

In addition, the state of New Jersey has allocated \$12 million to NSC over the same period.

Acquisition of a second Cyber 205 supercomputer last winter effectively

doubled the capacity of the center. This fall, the first of the class VII supercomputers, an ETA 10 with four processors, is being delivered to the site. Further upgrades will take place in 1988 with the installation of an eight-processor ETA 10 featuring 256 million words of memory.

For personal reasons, Joseph F. Traub said he is giving up his post as president of the CSC and professor of computer science at Princeton University to return to Columbia University as a professor and chairman of the computer science department.

CSC Chairman Richard G. Leahy said Traub's leadership has been the reason for the center's rapid expansion.

A committee has been appointed to find a successor.

Copyright Office to air separate registration

The US Copyright Office will conduct public hearings September 9 in Washington, DC, on whether computer display screens should be registered separately from computer programs.

USCO is especially inviting computer programming experts and professors of law and computer science to participate and comment.

The deadline to apply to appear at the hearings in person was August 28, but individuals interested in presenting written comments have until October 9 to do so.

The Copyright Office has registered computer programs as literary works since 1964. Following several recent court decisions concerning videogames, the office has also been registering pictorial and graphic screen displays as audiovisual works, independent of the computer programs that generate them.

USCO says that it doesn't currently register textual screen displays separately because there is no authorship in ideas or in the format or arrangement of text. Moreover, officials say, any literary authorship in the screen display would presumably be covered by the underlying computer program—itsself a literary work.

USCO is now getting an increasing number of claims to register textual and pictorial screen displays separately from the programs that generate them. It also noted that court decisions are split on the copyrightability of screen displays and lend no clear guidance on the issue.

Written comments submitted by mail should be addressed to the Library of Congress, Dept. 100, Washington, DC 20540. Hand-delivered comments should go to the Office of the General Counsel, Copyright Office, James Madison Memorial Bldg., Rm. 407, First and Independence Ave. SE, Washington, DC. In both cases, 10 copies should be included.

Additional information may be obtained by contacting Dorothy Schrader, General Counsel, Copyright Office, Library of Congress, Washington, DC 20559; (202) 287-8380.

IIA urges Congress to provide public access to scientific and technical information

Testifying before a Congressional committee, a representative of the Information Industry Association has urged the government not to abandon its responsibility for ensuring continuing public access to the nation's scientific and technical information.

Kenneth Allen, the IIA's senior vice president for government relations, appeared before the House Subcommittee on Science, Research, and Technology.

He said that the economic, technological and political strength of the United States depends on efficient and effective access to public information. This goal is best achieved through a partnership in which government and industry work together, Allen stated.

"The federal government has made a substantial investment in research and development and the creation of scientific and technical information," Smith said. "As citizens and taxpayers, we believe the government has a legitimate and appropriate interest in ensuring that citizens obtain full value from this investment."

Smith said that abolishing the National Technical Information Service or attempting to privatize it along lines that have been proposed would not be in the public interest because public access to federally funded scientific and technical information would be diminished.

Adoption of the IIA's proposal would retain a single repository for federally funded scientific and technical information, increase the public availability of this information by reducing its cost, restore Congressional oversight of this important function, and encourage the investment of private-sector capital to develop additional new products and services, he said.

The association also opposed a legislative proposal to reconstitute the NTIS as a wholly owned government corporation. Such a move would be counterproductive to the interests of the nation's scientific community in that it would discourage the investment of private sector capital needed to develop a diversity of information products and services responsive to the needs of the community, said Allen.

Scientists break the picosecond barrier with an electronic device

For the first time, researchers have broken the picosecond barrier using an electronic device.

In a project conducted by IBM scientists and partially supported by the US Office of Naval Research, electrical pulses have been produced lasting only half a picosecond, or 0.5 trillionths of a second.

As a result, the company's scientists will be better able to understand how electricity travels through components of computers, such as transistors, chip connections, and transmission lines. The experiment was called an important step in designing the ultrafast electronic computer components of the future.

Laboratory scientists used a laser and an ultrafast switch in the Yorktown Heights, New York, project. Pulses made timing the extremely brief events possible.

Today's fastest experimental silicon logic devices can switch on and off in about 30 picoseconds, while gallium arsenide devices can accomplish the same result in about 10 picoseconds, according to an IBM spokesperson.

However, to investigate the electrical behavior of these devices, researchers were challenged to measure pulses at least 10 times faster than these switching times.

The technique used to make the half-picosecond pulses can measure electrical pulses up to 20 times briefer than the switching times of the fastest present-day devices.

To generate the extremely short pulses needed, researchers fabricated a transmission line on a thin silicon layer. The transmission lines consisted of two parallel one-micron-wide aluminum strips two microns apart.

During operation, a voltage was maintained across the aluminum lines. A pulsed laser beam, consisting of a series of sub-picosecond light pulses, was split into two beams by a mirror. Because the beams followed different paths, it was possible to delay one light pulse stream.

Computer Society board urges repeal of US tax provision

The Board of Governors of the IEEE Computer Society has gone on record as being strongly opposed to a provision of US tax legislation passed last year that is considered detrimental to independent computer consultants.

Following six months of study and refinement by the Committee on Public Policy of the society's Membership and Information Activities Board, the CS Board of Governors has voted to join other societies and professional organizations in calling for the repeal of Section 1706 of the US Tax Reform Act of 1986.

Others that have taken similar action include the Data Processing Management Association, the Independent Computer Consultants Association, the National Association of Computer Consultant Brokers, and the Technical Consultants National Association.

Section 1706 makes it necessary for a computer consultant to, at the very least, maintain records to prove to the Internal Revenue Service that the type of consulting service he or she provided distinguishes the consultant as an "independent contractor" or "employee" under common law interpretation. In turn, each of these services is subject to different tax-deduction allowances.

The provision sets engineers, programmers, and draftsmen apart from other professionals. Were it to apply to a physician, for example, common law interpretation would conclude that:

(1) A physician performing a service in a hospital using hospital equipment is an employee of the hospital.

(2) A physician prescribing a painkiller or an antibiotic medicine to a patient is an employee of the patient.

(3) Only when a physician is examining a patient in the physician's office is he or she deemed to be working as an independent contractor.

To advise Congress of the society's stand, CS President Roy Russo dispatched the following letter July 14 to Rep. Dan Rostenkowski (D-IL) of the House Ways and Means Committee:

As president of the largest society of computer professionals in the United States, including some 68,000-plus US citizens, I was asked by the Board of Governors to inform you of the resolutions they passed unanimously on June 19, 1987, at a regular meeting held in Chicago. To wit,

RESOLVED: That the Board of Governors of the Computer Society of the IEEE requests that Section 1706 of the Tax Law of 1986 be repealed by the current Congress.

And

RESOLVED: That the Board of Governors of the Computer Society of the IEEE strongly recommends that Congress define the difference between an employee and an independent contractor.

These resolutions were precipitated by numerous complaints from society members that they had been singled out, as a class, for tax purposes. A subcommittee on public policy investigated the complaints and determined the cause as section 1706 of the 1986 Tax Reform Act. This section singles out computer professionals, many of whom (an estimated 10 percent) are independent contractors or consultants, and forces them to become employees for their livelihood or risk being in violation of the law.

The public policy subcommittee further reported on the haste in which Section 1706 was introduced by Senator Moynihan (D-NY) on the urging of lobbyists representing the National Technical Services Association, a trade group representing consulting organizations who employ their technical staff. No time for rebuttal by other responsible trade organizations was provided. There was no advance knowledge of any hearings held on

The ballot box

Computer wants to know how you, as a member of the Computer Society, feel about this issue and the stand the Board of Governors took on it.

If you favor the board's action, circle Reader Service Number 191; if you disagree with the board's stand, circle Reader Service Number 192.

We'll tally the votes and let you know the results in the November issue.

the amendment. No known opportunities existed for feedback from affected technical people before the law was enacted.

We are encouraged by the legislation introduced by Rep. Judd Gregg (R-NH) on January 29 (HR 792), which would postpone the effective date of Section 1706 of the 1986 Tax Reform Act for two years. This would give Congress two years to define the difference between an employee and an independent contractor for all professions. However, since Congress has been grappling with this problem since 1978, we do not see an early solution. Thus, we favor the Senate

version (S.491) introduced on February 5 by Senators Al D'Amato (R-NY) and Christopher Dodd (D-CT) to repeal Section 1706.

I respectfully request your support to obtain the repeal of Section 1706 and to have Congress define the difference between an employee and an independent contractor.

In a July 28 letter, Congressman Gregg thanked Russo for the society's support of HR 792.

"I am convinced that the two-year

delay would give the affected parties a chance to air their concerns, and Congress the opportunity to consider what changes may be in order," said Gregg, a member of the House Ways and Means Committee.

"Although we have numerous cosponsors, we still need additional support in order for the bill to be taken up in the Ways and Means Committee," he said, adding that he is working toward that end.

Technical committee report: Providing advanced security and privacy systems challenges the entire industry

*Carl E. Landwehr,
Chair, Technical Committee on Security and Privacy*

One of the chief questions confronting the computer industry today is: "How do we keep the hackers out?" If the only reading you do on the subject is what's in the newspapers, you might be led to think that this ought to be the primary goal of research into computer security and privacy.

But, the fact is, providing systems that meet real requirements for security and privacy poses a difficult and stimulating challenge that crosses many technical disciplines.

It is not enough for a system with security and privacy requirements to simply demonstrate that it can perform the functions demanded of it. It must also be possible to convince a third party that the system does not include hidden functions (e.g., Trojan horses, trapdoors) that threaten security.

Significant technical concerns in building systems that can provide security and privacy include:

- Clarity of specification and design,
- Application of the least-privilege principle (minimize the access rights of users and processes),
- Accountability for security-relevant operations,
- Enforcement of separation of duties (so no single individual can perpetrate fraud), and
- The sufficiency of testing and verification techniques.

Defining what the term "security" means for a given system is often accomplished by means of a model—an abstraction of the system that displays its properties from the security perspective. Usually, such models have been expressed informally in English or formally in the language of finite-state

machines or graph theory.

To be certain the implemented system has the security properties defined by its security model, you must show the correspondence between the implementation and the model. The means for demonstrating this correspondence can include informal inspection, testing, and advanced techniques for program verification.

These steps may sound straightforward, but they aren't. For example, security is often characterized informally as a system's ability to prevent the unauthorized disclosure or modification of information and unauthorized denial of system services. Yet many formal models deal only with the first of these properties; unfortunately, denial of service is rarely addressed.

The defense community has a long history of concern with security issues. This community has recently produced both a set of criteria for evaluating computer systems, according to its ability to enforce security policies, plus an organization to conduct such evaluations. As virtually all newspaper readers know, providing security and privacy is a growing problem for commercial and non-defense government systems as well.

Cryptography can be used to protect data against disclosures and detect their modification. Conventionally, encryption protects openly transmitted data (such as that transmitted over satellite pay TV channels) from unauthorized disclosure.

This is an appealing approach in that it can often be added to an existing, unsecured channel as an extra layer of protection with minimum disruption of existing operating methods. Of course,

with every encryption scheme comes a key distribution/key management problem.

Without the extensive software modifications required, finding ways to employ encryption to protect information stored in computers has proven difficult. A special processor called a trusted filter has been developed according to computer security principles to add encrypted checksums (called integrity locks) to records in an off-the-shelf database system. This represents one recent approach to combining encryption and other computer security techniques that is beginning to find practical application.

Providing security and privacy in computer/communication networks is an area of great concern. Increasing quantities of data, often sensitive or private, are transmitted electronically and stored in computers. Users of these services often seem unaware that such traffic is much more susceptible to interception and modification than the first-class mail service to which they are accustomed. TC members have been active in proposing schemes for digital signatures and other authentication methods to reduce these vulnerabilities.

The Security and Privacy TC is among the most active in the Computer Society. Each April, it sponsors a symposium to provide a forum for the presentation and discussion of new research and development findings in computer and communication security. Participation has been strong, and recent symposia have had to turn away some applicants. Selected symposium papers have appeared in a special issue of *Transactions on Software Engineering*, and best-paper prizes have been

awarded. Financial assistance will be available to students who present papers at the 1988 symposium.

In response to growing interest, the TC has formed a tutorials committee to help provide introductory materials to those new to the field, and has invited proposals for workshops devoted to such special topics as computer security foundations and database security.

Cipher, the TC newsletter, is published about four times a year.

The TC also cooperates with sister societies in related conferences and workshops, including the annual conference of the International Association for Cryptologic Research and the Aerospace Computer Security Conference.

This fall it will cooperate with ACM SIGSAC on a workshop dedicated to

investigating data integrity policies for automated information systems and with IFIP TC 11.6 on the Smart Card 2000 conference.

Membership in the TC is free; membership application forms are available from the Computer Society. For further information on the Security and Privacy TC, circle Reader Service Number 189.

Providing technical leadership is goal of Technical Committee on Microprocessors and Microcomputers

Martin Freeman,

Chair, Technical Committee on Microprocessors and Microcomputers

Unlike the disciplines of computer architecture, programming languages and operating systems, the microprocessor is not a discipline per se but a joining place for disciplines.

A microprocessor is a vehicle within which these disciplines operate; it thus acts as a catalyst for technical activity. For instance, reduced instruction set computer (RISC) technology which encompasses the aforementioned disciplines is exemplified by several RISC-based microprocessors that have recently become available.

In fact, today's microprocessors can deliver 10 million instructions per second (MIPS), providing several times the computing power of a VAX 11/780. Some engineering workstation companies are even projecting 100 MIPS workstations by the end of the decade, using improved versions of these microprocessors. Technology is forcing us to re-define our mainframe, mini-computer and microcomputer concepts.

The Technical Committee on Microprocessors and Microcomputers is helping in this regard by sponsoring conferences and vigorously pursuing the creation of standards. These activities are usually a cooperative effort among TCMM and various other IEEE TCs and ACM SIGs. TCMM sponsors the annual International Conference on Architectural Support for Programming Languages and Operating Systems (scheduled October 5-8, 1987) in conjunction with the TCs on VLSI and Operating Systems as well as ACM's SIGArch, SIGPlan, and SIGOps.

In the standards arena, our subcommittee, the Microprocessor Standards Committee (MSC), performs all of the coordination and management for our standards activities. During the past year, the MSC has been particularly active in standardizing VMEbus, Nubus, Multibus II and Futurebus, among others.

TCMM is not only interested in microprocessors per se, but also in I/O as a means to get information to and from the microprocessor. In order to broaden our efforts, work has begun in conjunction with the TC on Computer Languages to standardize the Modula-2 programming language and the Smalltalk programming language and environment. Recently, we sponsored the Summer 87 Futurebus Workshop, and we are currently a cooperating organization in COMPSTAN 88, the conference on computer standards. We also enjoy a unique relationship with *IEEE Micro* magazine; our standards director writes a column that deals with progress being made through our standards activities.

While the construction of microprocessors is important, their use in applications is equally important. High-speed microprocessors breed high-performance systems. Transaction processing is one application area where this increase power can be utilized. TCMM, along with the TCs on Data

Engineering and VLSI, will sponsor a Workshop on Transaction Machine Architectures next year. The objective of this workshop is to help develop the field of transaction processing through consideration of high performance systems that microprocessor technology has made possible.

In the past, we published a short annual newsletter. We will soon launch a more substantial offering, with articles based on our conference and workshop activities as well as our standards efforts.

TCMM's objective is to provide technical leadership to members of the Computer Society and others in the technical community through its standards activities and its commitment to developing areas of joint interest with its associated technical committees and special interest groups, both within the society and outside it.

Application forms to join TCMM and additional information about it may be obtained by circling Reader Service Number 188.

Moving?

PLEASE NOTIFY
US 4 WEEKS
IN ADVANCE

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

ATTACH
LABEL
HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address below.
- If you have a question about your subscription, place label here and clip this form to your letter.

Name (Please Print)

New Address

City

State/Country

Zip

Eniac's historical significance observed

The Computer Society will co-host ceremonies at the University of Pennsylvania in Philadelphia September 22 commemorating development of the Eniac.

The forerunner of today's large electronic digital computers, Eniac was developed and constructed in the mid-1940s at the Moore School of Electrical Engineering on the campus.

A plaque recognizing Eniac as an IEEE electrical engineering milestone will be dedicated at 7 p.m. at the Towne Building and Alumni Hall.

On hand from the society will be Kenneth R. Anderson, second vice president for technical activities.

Anderson, IEEE President Henry I. Bachman, and the university's Joseph Bordogna will deliver dedication remarks.

Also hosting the event will be the Philadelphia Section of the IEEE and the Unisys Corp., which traces its roots to Eniac's development.

IEEE members and computer industry representatives are invited to attend.

Computer science programs granted accreditation

Twenty-five additional four-year computer science programs at US colleges and universities have been accredited by the Computer Science Accreditation Commission of the Computing Sciences Accreditation Board.

Addition of the 25 programs brings the total now listed by the CSAC/CSAB to 48.

The newly accredited baccalaureate programs are those of American University, Auburn University, Ball State University, Baylor University, California State University at Chico, California State University at Northridge, Canisius College, Eastern Washington University, Fairleigh Dickinson University, Florida State University, George Washington University, the University of Houston, Lehigh University, the University of New Hampshire, the University of New Orleans, North Carolina State University, the University of

North Dakota, Northeast Louisiana University, the University of North Florida, the University of Southern Mississippi, the University of Southwestern Louisiana, State University of New York at Albany, the US Naval Academy, Western Washington University, and Wright State University.

The CSAB bases its decisions on evaluations of faculty, curriculum, laboratory and computing resources, students, and institutional support.

Thirty-five institutions had applied for accreditation during the current 1986-87 cycle. This was the CSAB's second series of accreditations since it was established by the Computer Society and the ACM in 1984.

The CSAC expects to evaluate another 20 to 30 programs during the 1987-88 cycle, beginning this fall with campus investigations and interviews with faculty members and students.

Interruption in magazine service

During the period April through August of 1986, the Computer Society experimented with an alternate method of surface delivery of magazines outside the US, affecting the May through September issues. The new carrier, which was retained in an attempt to improve service, in fact experienced sporadic shipping delays in two postal gateways, resulting in late deliveries for some members in Regions 8-10.

Reports of nondelivery began showing up in August. As soon as the source of the problem was identified, the alternate service was discontinued and the former method of shipment (via US Mail) was restored, commencing with the October issues of *Computer*,

CG&A, *D&T*, and *Micro*, the November issue of *Software*, and the Fall issue of *Expert*. Approximately 2000 complaints of late delivery or nondelivery have been received by the society's Publications Office since last August, reaching a peak of 500 during the month of November and dwindling to zero last May.

Any remaining claims for these or other missing issues should be addressed to the Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720. Back issues will be shipped so long as inventory holds out.

The Computer Society apologizes for the inconvenience this has caused to some of our members.

Students invited to enter computer bridge contest

The Computer Society is sponsoring its first computer bridge contest for college students and is inviting all interested students to develop bridge-playing programs so they can compete in a tournament.

The winning team will be awarded a personal computer.

The event has been tentatively scheduled for March 1, 1988, in San Francisco, although the date and place may be changed if too few teams appear ready to compete by that time, according to contest organizer Willis King of the University of Houston.

All student teams sponsored by society student chapters are eligible to

enter, and special rules and regulations will govern the event, King indicated.

Each player in a given game will be simulated on an independent computer. Therefore, each team will have to have at least two computers and, preferably, four so that no particular security measure will be needed when hands are to be replayed after switching positions (from NS to EW and vice versa).

Specific rules will govern bidding systems, bidding conventions, psyching, lead and carding styles, scoring, response times, penalties, etc.

For further information about the bridge contest or about forming student chapters, please circle Reader Service Number 190.

**Late Magazines?
No Magazines?
Membership
Status Problems?
No Answers
To Your
Complaints?**

**Let your
Computer
Society
Ombudsman
cut
through
the red
tape
for you.**

Computer Society
Ombudsman
IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720



NEW PRODUCT REVIEWS

Editor: Richard Eckhouse, MOCO, Inc., P.O. Box A, 91 Surfside Rd., Scituate, MA 02055; Compmail+, r.eckhouse

Interactive videodisc technology

Sorel Reisman

California State University, Fullerton

The need to use time efficiently, keep costs down, and ensure mastery of material being taught have forced industrial training departments to explore different instructional techniques to meet their objectives. These requirements have paved the way for the use of new technologies in industrial training as well as in elementary, secondary, and postsecondary education. Examples have included overhead projectors, instructional television, group and individual use of video cassette recorders, computer managed and assisted instruction, and more recently, computer-based interactive videodisc.

Overview of the videodisc industry

The videodisc industry began in about 1977 when the first laser optical prototype players were made available to videodisc software developers. Realization of the initial forecasts for the consumer videodisc market proved disappointing. Many companies with inferior videodisc formats abandoned the market and yielded to what has become the one surviving standard—laser optical videodisc.

While it may be argued that in terms of consumer volumes industry growth has been slow, since the early 1980s there has existed a small group of committed entrepreneurs who believe in the technology. This group continued to develop, market, and otherwise pioneer the major technical advances that kept the industry economically afloat.

Among other things, these small companies provided a variety of incompatible hardware components, authoring systems, and related miscellaneous products, all of which served the needs of customers willing to build their own integrated systems.

More recently major corporations have reentered the market with completely integrated interactive videodisc products. Their entry and their products have given the industry significant impetus. The availability of such systems from reputable and well-known

manufacturers greatly reduces a training department's risk of encountering hardware or software problems. A casual examination of the prices of the new systems reveals that they appear to be more expensive than the "do-it-yourself" systems, since they cost, on average, \$8000. However, the increased reliability of the new products, and the time saved by attending to the objectives of a videodisc project rather than the problems of system maintenance, result in greater cost savings in the long run.

Most noteworthy among the integrated systems are Sony's View System and IBM's new InfoWindow System. Another new product is EIDS (Electronic Information Delivery System), Matrox Electronic Systems' response to a US Army Request for Proposal for a standard integrated interactive videodisc training system. With products such as these, after almost ten years of cottage-industry-level growth, the optical videodisc industry is finally on the verge of meeting its original forecasts.

Optical videodiscs

A complete interactive videodisc system has the following components:

- (1) videodisc,
- (2) videodisc player,
- (3) display screen,
- (4) interface hardware,
- (5) computer with software, and
- (6) audio system.

A more detailed description of some of these items is provided below. Some of the hardware devices in this list may also include other related peripherals such as keyboards, touch sensitive screens, etc.

Videodiscs. A videodisc is a two-sided, eight- or twelve-inch diameter reflective plastic disk with a clear plastic coating to protect the underlying surface from dirt, light scratches, grease, etc. The underlying surface contains billions of micropits, which can be encoded to contain audio, video, and digital data.

Micropits are organized along 54,000 concentric tracks on each side of the disk. This structure, using the constant angular velocity (CAV) encoding scheme, provides a maximum capacity

of 54,000 independently addressable color images, or 30 minutes of continuous video motion (at 30 frames per second), or a mixture. Video motion may be accompanied by two independent sound tracks. In addition to these audio-visual "data," conventional computer data, equivalent to thousands of floppy disks, may also be stored on the videodisc.

One of the major advantages of the CAV encoding scheme is the ability to mix all of these data types on one side of a videodisc and to retrieve them randomly as required.

A second data encoding scheme, called constant linear velocity (CLV), provides up to sixty minutes of continuous video motion, accompanied by the two audio tracks. This format does not provide as much flexibility for mixing of data types, or for random data retrieval.

CAV discs are usually used for interactive sessions; CLV discs are better suited for the continuous play requirements of consumer-oriented prerecorded films.

The videodisc player. A videodisc rotates at 1800 revolutions per minute while a low-power laser reads its data. Industrial videodisc players can access any of the 54,000 (CAV recorded) tracks in under three seconds.

Videodisc players contain a microprocessor-based controller to control the laser and other electromechanical operations, and for data encoding. Users employ the microprocessor when they wish to branch to particular frame addresses (or to optionally encoded "picture stops" and "chapter stops") encoded on the videodisc. Control functions include

- play,
- reverse,
- stop,
- pause,
- index frame display,
- scan forward/reverse,
- search,
- slow forward/reverse,
- still forward/reverse, and
- toggle audio channel one and/or two.

All videodisc players can play either CAV or CLV recorded discs. Except for the audio features, most of the functions listed above are not available when using a CLV (continuous play) disc.



The Sony View System consists of the SMC-2000 microcomputer and the LDP-2000 videodisc player.

Interface hardware. Interface hardware consists of a variety of cables and printed circuit boards that enable the electronic integration of a computer, videodisc player, and display. Software that executes in the computer controls one or more videodisc players, which must be attached to the computer. Although proprietary interfaces exist, the most common are the standard RS-232-C and IEEE-488.

Another necessary interface contains the output display control circuitry for computer text and graphics, videodisc output, or the simultaneous display of both.

Because of the infinite combinations of function definable with the integration of the various hardware components, a large number of manufacturers offer interface products. In many cases, the extreme variability of interface function has caused (allowed) manufacturers to design videodisc authoring languages specifically for their proprietary interfaces. Until the recent availability of "reputable" integrated systems, this infinite wealth of function provoked a

great deal of confusion in the marketplace and to a large degree inhibited more widespread use of the technology.

Other peripherals. Additional devices are often integrated into an interactive videodisc system. The list includes

- touch sensitive screen,
- light pen,
- digitized audio output,
- speech recognition,
- digitized imaging, and
- multiple videodisc player control.

Many of these devices can be found as standard features or as options in some of the emerging integrated system products.

Other optical laser formats. Today, the only viable disc standards are those based on the reflection of laser light from the surface of an optical disc. Other disc standards that tend to receive significant attention are all based on this technology. Their essential differences are their physically smaller size and the manner in which data are encoded on them.

Other related optical disc product technologies concern the end user's ability to record or rerecord data on a videodisc. Industry experts agree that, as desirable as this feature may be, it will not be generally available (cost effective) for quite some time. Available products, which are fairly expensive, tend to be described by such acronyms as DRAW (direct read after write) or WORM (write once, read mostly or many times).

Interactivity and videodiscs. Historically, professional curriculum designers have generated definitions of interactivity and its relationship to videodisc technology. They have also spearheaded the slow but continued development of videodisc applications. The generally accepted classification scheme described below is based upon instructional applications of the technology and the levels of "intelligence" available from different hardware configurations. Classification levels 0 through 4 refer to increasingly rich interactive sessions available with the technology.

Level 0. In this level, random access is not used. A disc plays from beginning to end without stop. Discs designed for this use are usually encoded in the CLV (sixty minutes of play per side) format. Except for stereo sound, none of the special videodisc player features (freeze frame, slow forward/reverse, etc.) is available.

Level 1. With Level 1 systems, the user does not play the (CAV) disc straight through. Instead, the user explicitly causes the player to branch to different disc addresses. User interaction is facilitated by the use of picture and chapter stops encoded on the disc. These are "breakpoints" on the disc at which the player automatically stops and displays a still video image. These picture/chapter stops are usually designed to display a menu of user-selectable options. The user may key in a menu choice through a remote control pad, causing the player to branch to a frame address containing material associated with the selected item.

Level 2. In Level 2 systems, the videodisc player's internal microprocessor controls learner interaction. A digital control program stored on the disc is loaded into the player's microprocessor at start-up time. Because videodisc player controllers are relatively primitive, only a low degree of interactivity can be preprogrammed. Level 2 applications are more difficult to design than Level 1 applications. The control programs must be encoded on the disc and cannot be altered. The programming languages used are usually specific to a

manufacturer's Level 2 videodisc player products.

Level 3. In a Level 3 system, an external computer is attached to the videodisc player. The user interacts with the videodisc-based material through computer input devices (such as a keyboard, touch sensitive screen, etc.) rather than through the player's remote controller. The control program is computer-resident rather than player-resident and, unlike the programs of Level 2 systems, can be modified and reprogrammed.

Level 4. Within the videodisc industry there exists no agreement on a specific definition of Level 4 systems. Some define Level 4 configurations as those that contain a computer-resident program that can generate the following outputs on a single output display:

- (1) video and still images—from the disc player,
- (2) computer text and color graphics—from the computer, or
- (3) a mixture of both of the above—logically controlled from the computer program through the interface hardware.

Videodisc manufacturing

One of the early problems facing the laser optical disc industry was disc manufacturers' inability to manufacture or replicate videodiscs at a reasonable cost. The process first requires that a master videotape be produced. The tape is then used to "cut" a master videodisc.

To avoid introducing impurities into the master disc as well as the replications, a clean environment is essential. In the early 1980s, this was not well understood and resulted in the production of many imperfect discs. Manifestations of the impurities included random frame skipping, frame sticking, or audio hiss. Since then, problems of playability have largely been solved, with an associated decrease in both the complexity of the process as well as the related costs of manufacturing.

Application design

A weakness of many interactive video programs, and of most application programs that involve interactivity, is poor design. In instruction, the primary application for interactive videodisc, instructional systems design is a fairly



An IBM InfoWindow System can include the InfoWindow display, an IBM PC, and a non-IBM videodisc player.

well-developed art with many texts covering the methods. One individual rarely possesses all the skills necessary to produce and use effective conventional instructional materials. Because interactive videodisc design involves a confluence of instruction, computer programming, and television production, the task becomes even more complex. The necessary skills include

- instructional systems design,
- subject-matter expertise,
- writing,
- text design and layout,
- photography,
- graphics design,
- television production,
- computer programming,
- project management,
- program evaluation, and
- implementation.

The complexity of producing interactive video material is so widely recognized that an entire industry has developed to support the process. Early efforts to formalize the process specified a need for complex hardware, soft-

ware, and procedures. Significant price and performance improvements in technology have eliminated the need for special hardware and allowed practitioners to develop procedures that seem to work.

Authoring languages and systems.

Authoring languages and systems used with videodisc players are designed to be used with specific computers, videodisc players, or integrated videodisc systems. Because they have been designed for different hardware, they usually provide different author-selectable functions to control the different hardware features. While uses of systems and languages differ, they tend to be conceptually similar. In general, they have many features similar to conventional computer-assisted instruction authoring systems. However, a major difference is that they enable the author to use audiovisual material as the primary focus of his or her program.

Authoring software is usually available in at least three levels of complexity:

(1) Authoring languages. These languages are for experienced computer users who prefer to develop material in a manner similar to that of a computer programmer who designs computer applications such as payroll systems.

(2) Menu-driven systems. These systems guide the author through various programming possibilities based upon menu options and provide opportunities to make decisions about the design of the instructional material. Menu-driven systems generate high-level code, such as that used in authoring languages. It is sometimes possible to break out of the menu scheme and write customized control modules using a programming or authoring language.

(3) Application systems. These systems have the characteristics of menu-driven systems, but the menu schemes and decision opportunities presented to the author ensure the generation of computer code specific to a particular application.

The IBM InfoWindow hardware is supported by authoring software that represents each of these classifications of language or system. Pilot is a well known computer-assisted instruction language modified to support the videodisc function of the InfoWindow. LSI, IBM's menu-driven authoring system, can be used to code interactive instruction or video-based product presentations in public kiosks.

Examples of application systems show up in both Sony's and IBM's systems; both products have been used in interactive information kiosks at the World's Fair and Disneyworld.

Current systems. In general, all of the systems currently marketed are functionally equivalent to one another. Each system consists of a personal computer attached to a videodisc player with an RS-232-C or IEEE-488 interface. An end user interacts with the hardware through a keyboard, joystick, lightpen, or touch sensitive screen. Output is audio from the disc or from a digitized audio device, video stills or motion from the disc, text and graphics from the computer, or a combination of video and text and graphics superimposed on one another.

An example is EIDS, the Ada of integrated videodisc systems. Its features simply reflect the US Army's apparent strategy of requiring proprietary software or hardware when other equally good products are available.

Both the IBM and Sony systems reflect an eight-year competition between the two companies. Each company manufactures a videodisc product

line more or less functionally comparable with the other. They both adhere to agreed upon audio and video industry standards, hence both play industry standard optical videodiscs.

By 1981, when IBM announced the IBM PC, videodisc aficionados had already been attaching Apple II's, Ataris, Commodores, and other inexpensive micros to videodisc players. The relatively late arrival of the IBM PC in the marketplace, and the popularity of the Apple II relative to the other micros, seemed to have predestined the development of Level 3 material for Apple II's.

With the popularity of the PC, it was only natural for IBM to introduce the InfoWindow System, based on the IBM PC de facto industry standard and the Pioneer technology with which IBM had become familiar through an earlier joint venture with Pioneer. The first InfoWindow System components consisted of an IBM PC-XT or PC-AT, EGA-based graphics and display (outfitted with a touch sensitive panel), and interface boards and software drivers for Pioneer videodisc players.

Since April 1987, IBM has extended the InfoWindow System to include support by the new PS/2 series, and recently, the Sony videodisc player product line. One can only surmise that the inclusion of Sony players results from an arrangement between Sony and IBM in which Sony was forced to concede that US corporate customers prefer IBM micros over most Japanese near-clones. And in any case, it is advantageous for IBM to support both of the popular videodisc player product lines.

Selection of a system. Despite the numerous combinations of products and peripherals available, the selection of an integrated videodisc system is quite straightforward. Apple-based systems are still available—not from Apple Corporation, but from system integrators who use either Pioneer or Sony players. The limitations of these systems are a function of the computer's features (speed, graphics, etc.) rather than the videodisc player. These systems are usually the least expensive integrated systems.

IBM and other PC-based integrated systems are also available from system integrators. These systems use Pioneer or Sony players and usually cost less than the Sony View or IBM InfoWindow System because they use PC compatibles.

The last classification, and the most expensive, is the fully integrated system

manufactured by one of the major corporations. In general, they will cost the most, but the price ensures the long-term support of the manufacturer. One problem to consider, at least in the sense of Sony versus IBM, is the different orientations of the two companies.

In a simple sense, IBM is a computer company and Sony is an audio/video company. The Sony View System has somewhat more powerful video features than the Pioneer players that IBM first supported. On the other hand, IBM's computer support is far superior to Sony's. It is likely that IBM will continue to improve its computer support, and Sony will do likewise with its video products. InfoWindow owners are therefore more likely to benefit from new computer advances, while Sony View owners will benefit from new video advances.

The choice you make will depend both on your own orientation and that of your videodisc projects. If your projects are designed to use videodiscs as extended databases, then the computer dimension is most important. On the other hand, if your projects are designed primarily around the audio-visual medium, then you should focus upon that dimension. In either case, your task will be challenging, rewarding, and most of all, fun.

Bibliography

- L.J. Briggs, ed., *Instructional Design*, Educational Technology Publications, Englewood Cliffs, N.J., 1977.
- M. DeBloois, K.C. Maki, and A.F. Hall, *Effectiveness of Interactive Videodisc Training: A Comprehensive Review*, The Monitor Report Series, Future Systems Incorporated, The Videodisc Monitor, Falls Church, Va., 1984.
- F.T. Hofstetter, "Perspectives On a Decade of Computer-Based Instruction, 1974-84," *J. of Computer-Based Instruction*, Vol. 12, No. 1, Winter 1985, pp. 1-7.
- N.V. Lippa, *A Practical Guide to Interactive Video*, Knowledge Industry Publications, White Plains, N.Y., 1984.
- E. Schwartz, *The Educators' Handbook to Interactive Videodisc*, Assoc. for Educational Communications & Technology, Washington, D.C., 1985.
- Videodiscs for Education: A Directory*, Minnesota Educational Computing Corp. (MECC), Minneapolis, Minn., 1986.

NEW PRODUCTS

Edge supermini delivers RISC performance with CISC instruction set

Edge Computer has announced the Edge 1000 family of superminicomputers, targeted at value-added resellers and systems integrators using the Motorola 68010/20 microprocessors.

According to the company, the new processors are 32-bit superminicomputers implementing the Motorola instruction set in the Edge mainframe architecture. This reportedly delivers RISC-like performance through Motorola's CISC-based 68000 family instruction set.

The dual-processor Edge 1200 with 8M bytes of system memory, 337M bytes of hard disk storage, and the GSX Unix operating system costs around \$128,000. It reputedly achieves 11 million instructions per second. The single-processor Edge 1100 in the same configuration costs \$104,000. It reputedly achieves 6 MIPS.

The 1000 family reputedly features an average instruction time of 1.59 cycles per instruction, with memory boards using dual surface mount technology for high density. The 32-bit processor is a VLSI implementation of a Harvard supercomputer architecture that simul-

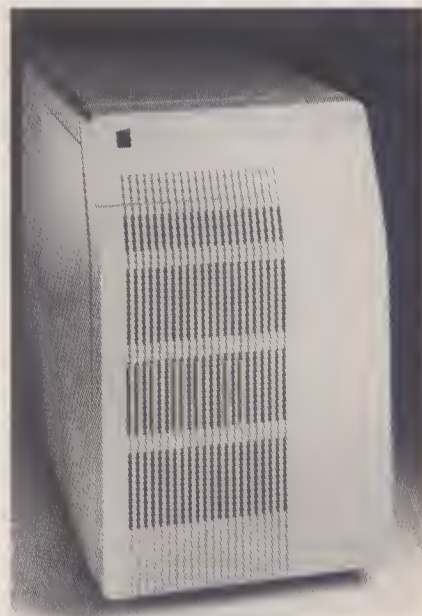
taneously fetches instructions and operands over multiple 32-bit buses. The dual four-state pipelined processor overlaps the execution of multiple instructions.

The CPU has two independent, decoupled pipelines: the instruction fetch pipeline and the operand executive pipeline. IFP fetches instructions and OEP decodes and executes them. Pipeline performance relies on the branch prediction unit, which consists of a 4096-entry branch cache and a 16-entry last-in/first-out return stack.

The floating-point accelerator conforms to the IEEE standard 754, draft 10.0, and supports 32-bit and 64-bit floating-point formats and 8-, 16-, and 32-bit signed and unsigned integer formats.

The Edge 1000 family offers a range of industry standards, including VMEbus and Multibus, SMD for 8-inch drives, AT&T Unix System V, Ethernet, TCP/IP, Sun NFS, and SNA. Languages include ANSI Fortran 77, C, Cobol, and ISO Pascal.

Reader Service 30



Both the Edge 1100 single-processor and the Edge 1200 dual-processor models fit into a 17-inch wide, 29-inch high cabinet that includes peripherals and up to 64M bytes of memory.

Anza coprocessor brings neurocomputing to personal computers

Hecht-Nielsen Neurocomputer offers the Anza Neurocomputer coprocessor board for the IBM PC-AT and compatibles. The company usually bundles the board with a Zenith Model 248 PC. The Anza uses a Motorola MC68020 CPU and MC68881 floating-point coprocessor coupled with 4M bytes of one-wait-state dynamic RAM.

The company's newest model, the ANZA 386 Neurocomputing System, AZ1500, has a Zenith 386-80 host equipped with the Intel 80386 16-MHz microprocessor, 1M byte of memory, 80M-byte hard disk storage, EGA adaptor, and 13-inch EGA color monitor. The AZ1500 costs \$19,500.

According to the company, the Anza can implement neural networks containing up to 30,000 processing elements (neurons) with up to 480,000 interconnects.

Software supplied includes the User Interface Subroutine Library and five Basic Network Packages. The UISL is a collection of instructions that provide access to Anza functions from within programs written in C, Pascal, Fortran, and Basic. Each of the network packages is a user-configurable implementation of a network paradigm specifying the interconnection structure of a network and the form of the differential equations that determine the behavior of individual processing elements.

The Anza costs \$9500 for the board alone; \$14,950 bundled with a 10-MHz Zenith Z-248 PC with a 20M-byte hard disk; or \$18,950 with a Zenith Z-248 PC with a 40M-byte hard disk, 1M byte of extended memory, and an 80287 coprocessor. Both Zenith systems include a 1.2M-byte floppy drive, EGA color board, and monitor.

The company also offers three training courses. The Anza User's Course is a two-day course for people already familiar with neural network technology. It costs \$595. The Introductory Neurocomputer Applications Short Course is a five-day course for those not yet familiar with neural networks who wish to apply the technology to real-world problems. It costs \$1950, \$950 if an order for Anza equipment is placed simultaneously. The Advanced Neurocomputer Applications Course is a 30-day hands-on course in the applications of neural network technology. It costs \$15,000.

Contact the company for specifics on the courses offered.

Anza: Reader Service 31
Courses: Reader Service 32

Operating systems extend MS-DOS

AI Architects has announced OS/286 and OS/386, protected-mode extensions to MS-DOS that reputedly break the 640K barrier for IBM PC-AT class machines and enable 32-bit performance on 80386 machines. The new operating systems initially target applications developers.

According to the company, the systems work with DOS to provide improvements in three areas. First, they eliminate memory constraints, so that programs can address all extended memory installed, up to 15M bytes for the 80286 or 4G bytes for the 80386. Second, more terminate-and-stay-resident programs can coexist with a large application running in protected mode. Third, performance in 80386 systems improves with the shift from 16-bit to 32-bit operation.

Other features include support for existing DOS calls; new INT-21 calls for manipulating segments, invoking real-mode routines and interrupt handlers, and addressing physical memory directly; interrupt vector support; debugging; and the ability to run non-Windows programs in a window.

The development toolkit includes the kernel and linker, debugger, and command processor. The toolkit costs \$495.

Options include 16-bit and 32-bit compilers; High C, Professional Pascal, or F77L Fortran; a 32-bit assembler, and the 386 Hummingboard. The 32-bit compilers cost \$895.

The company also offers runtime licenses for OS/286 and OS/386.

Reader Service 33

Software design tool employs graphics of IBM PS/2

Visual Software has announced a computer-aided software engineering design tool called vsDesigner. According to the company, the product takes advantage of the enhanced graphics capability of the IBM Personal System/2 by offering native compatibility with VGA, the advanced graphics device driver of the PS/2.

Visual Software calls vsDesigner an automated drawing and documentation tool that uses a multiuser, object-oriented database. Designers can reportedly transfer information from a personal computer to a mainframe over a local-area network such as the IBM Token Ring and PC Net, and Novell and 3Com LANs.

Programmers can use Yourdan and Warnier-Orr methodologies, or their own customized symbols. They can choose cursor or mouse, screen or menu formats and interface protocols, and user interface for the word processor.

The software provides a drawing editor, virtual display, integral word processor, attribute editor, and report generator. According to the company, the following functions are performed

in an integrated environment: diagramming; capture of associated text for requirements, definitions, and design specifications; central storage of documentation and program information (encyclopedia); project management; and quality assurance, tracking, and reporting.

The product reputedly has an open architecture. It also executes on the IBM PC, PC-XT, PC-AT, and compatibles with color support in a stand-alone mode or in a multiuser LAN.

The company offers on-line updates through an electronic bulletin board. Users need an encryption device to plug into their computers.

In single units, vsDesigner costs \$7500. For 15 or more systems, the price drops to \$4600.

The company also offers a lease/rental program and training. The maintenance agreement includes updates through the electronic bulletin board, for \$1150 per unit. A demonstration version of the program costs \$95.

Reader Service 34

Concurrent connects PCs to Series 3200 superminis

Concurrent Computer has announced the PENnet PC software package for connecting IBM PCs to the Concurrent Series 3200 superminicomputer systems.

PENnet PC gives users access to PENnet Plus networking facilities. Used together, the two products provide PC users with terminal emulation, file transfer, programming interface,

and access to Reliance Office automation tools to obtain data or files from a local Series 3200 system and share information with other users.

PENnet PC requires an IBM PC, PC-XT, PC-AT, or compatible. It costs \$2500 for a site license.

Reader Service 35

Fujitsu tackles finite element analysis on a PC

Fujitsu America has announced the Engineering Library for Modeling, or Elm, for PC-based finite element analysis. The package consists of ElmAnalysis and integrated graphics pre- and postprocessors.

The preprocessor, ElmPrelude, employs an interactive graphic approach to creating structural models. It combines pull-down menus, icons, dialog boxes, and on-line help. A mouse reduces keyboard data entry.

ElmPrelude also employs a user-defined grid system and automatic input of finite element analysis property values, based on embedded engineering library files and user-supplied data.

ElmAnalysis for structural engineer-

ing performs static, modal (eigenvalue), and earthquake analysis (response spectrum analysis). The element library includes three-dimensional beam elements, truss elements, triangular and quadrilateral shell (plate) elements, and two-dimensional four-node and eight-node isoparametric elements.

ElmAnalysis provides a free-format input scheme for users who wish to perform batch-mode data entry rather than use ElmPrelude.

ElmEpilog, a graphics postprocessor with the same user interface as ElmPrelude, allows users to review and manipulate the output of ElmAnalysis. It provides visual displays and printouts or plots of the analyzed structure's

undeformed shape (model geometry), deformed shape (static analysis results), and mode shapes (eigenvalue analysis results).

Elm requires an IBM PC-AT, PC-XT, or compatible, math coprocessor, 10M-byte hard disk, Microsoft-compatible serial mouse, EGA graphics board and monitor, one 360K-byte floppy disk drive, minimum 512K-byte memory, and MS-DOS 3.0 or higher.

The complete package, including ElmPrelude, ElmAnalysis, and ElmEpilog, costs \$3990. A two-dimensional Elm package costs \$495.

Reader Service 36

WISC blends RISC and CISC architectures

Mountain View Press distributes a product called WISC, for Writable Instruction Set Computer. WISC, developed by WISC Technologies, reputedly blends reduced instruction set computer and complex instruction set computer architectures.

The WISC CPU/16 plugs into an IBM PC, PC-XT, PC-AT, or compatible. It includes a processor board and a memory board. Microcode is written with the microassembler and loaded from the host along with the application program.

The 16-bit coprocessor board directly executes high-level stack-oriented programs. It contains 74 standard integrated circuits with 120-nanosecond

static random access memories for microcode and program memory. The host interface permits altering registers and memory.

The complete package, for \$1500, includes assembled and tested boards, processor software, number package, microcode assembler, cross compiler, diagnostic programs, source code, and documentation.

A wirewrap kit costs \$900. The manual alone costs \$50. A collection of papers about WISC costs \$15.

A 32-bit version is under development.

Reader Service 37

Communications server brings PCs into LANs

Bridge Communications has announced a board-level PC communications server called Personal Communications Server/1.

According to the company, PCS/1 provides local-area network capabilities that allow IBM PCs and compatibles to access PC LAN file servers and print servers, perform terminal emulation and high-speed file transfers, run networked application software, and take part in network management within a LAN environment.

PCS/1 includes PC software and an intelligent network controller board. It implements the Transmission Control Protocol/Internet Protocol network protocols and is compatible with other TCP/IP LAN products. It was designed for use with Ethernet (IEEE 802.3) baseband or Bridge 5M-bit-per-second broadband networks.

Asynchronous terminal emulation is provided by the PCS/1 TTY terminal emulator or through third-party terminal-emulation programs through a Bridge application program interface. Support of multiple simultaneous sessions allows the user to switch back and forth between hosts. The asynchronous

file-transfer capability of terminal-emulation products can also be used with the PCS/1.

Compatibility with Microsoft Windows permits asynchronous connections, file-transfer sessions, and other applications running in separate windows on the PCS/1 screen.

According to the company, the PCS/1 relies on the Intelligent LAN Adapter (Ilana) board, a TCP/IP network protocol processor that provides direct LAN attachment for IBM PC-AT or PC-XT machines, as well as for IBM's PS/2 Model 30. An on-board MC68000 CPU coupled with an Intel 82586 network coprocessor chip reputedly lets the Ilana board act as a front-end processor for the PC. A 512K-byte memory stores software and provides communication buffers.

The PCS/1 costs \$1195. It consists of a software engine (TCP/IP protocols, Telnet, FTP, Network Management User Interface, TTY terminal emulator, and Bridge Application Program Interface) and either the Ilana/AT or Ilana/XT board.

Reader Service 38

FS-386 slotcard system costs under \$3000

Fivestar Computers offers the FS-386 slotcard computer system with 12 expansion slots and 6 peripheral drive slots, and a 238WT power supply.

The motherboard containing the Intel 80386 CPU is inserted into a 16-bit slot. Speeds are software switchable between 6 and 16 MHz. The base system also

includes a 1.2M-byte or 360K-byte floppy disk drive, one serial port, one parallel port, keyboard, clock/calendar with battery backup, DOS, Basic, and operations manual. It is available with 2, 4, 8, 10, or 16M bytes of RAM.

The FS-386 costs \$2995.

Reader Service 39

Board connects PS/2 with System 34/36/38

IDEAssociates has announced a 5251 emulation board that connects IBM Personal System/2 models 50 and 60 with IBM System 34/36/38 minicomputers. IDEAcomm 5251/MC reputedly enables PCs to emulate IBM 3180, 5251 model 11, 5291, and 5292 model 2 terminals and can be used with color or monochrome monitors.

IDEAcomm 5251/MC also permits PC parallel or serial printers to emulate IBM 5224, 5225, and 5226 printers as well as the 5219 letter-quality printer.

The product supports up to four concurrent host sessions, viewable simultaneously by using a windowing feature.

IDEAcomm 5251/MC emulation and software cost \$895.

Reader Service 40

Apollo integrates PCs with workstations

Apollo Computer has expanded its line of products integrating personal computers with workstations.

The Domain/PC Emulator software package runs PC applications on a Domain workstation, reputedly without requiring additional hardware. It simulates an IBM PC-AT and is compatible with all Domain workstations. It costs \$500.

The Domain/PCI-Ring hardware/software communications package connects a single PC to the Apollo Token Ring network. It includes communications software and a network controller, a single board that fits into an expansion slot in the IBM PC-XT, PC-AT, and compatibles. It includes a D-sub connector that interfaces to the token ring network through Apollo's Domain/DQC Quick Connect. The connector can also be adapted for a BNC interface to the ring. It requires a PC with at least 256K bytes of RAM running MS-DOS 3.1.

The company plans to release a similar product in 1988, Domain/PCI-Enet, which will connect a single PC to an Ethernet network.

Emulator: Reader Service 41

PCI-Ring: Reader Service 42

Image processing workstation combines with Sun-3

Vicom Systems has announced the second model in its Vicom-VME image processing workstations. The new workstation is reportedly designed as a peripheral processor to a Sun-3 workstation.

The Vicom-VME offers separate modules for image acquisition, processing, and display. This permits customizing each workstation to meet specific needs, according to the company.

The Vicom-VME microcomputer system boards are connected to the VME-bus, whose address and data lines and many of the control lines are connected to the Vicabus. The Vicabus is a set of five image buses to interconnect image memory, dedicated processors, and other image-processing elements of the system.

The company offers several types of video rate data acquisition systems. These digitizers reside on the Vicabus and can accept input from cameras,

VCRs, CCDs, and slow scan devices. Digital and analog inputs are supported.

The pipeline image processor does high-speed image processing. It includes a pipeline controller, point processor, and hardware convolver.

The display system provides buffer memories, cursor and display generators in various combinations to produce up to six monochrome or two full-color displays in the 512 × 512 format.

According to the company, the implementation of Unix 4.2 SunOS as the operating system provides the user with C, Fortran 77, Pascal, and other higher level languages for algorithm development and customized processor control.

Prices for the new model Vicom-VME begin at \$48,000.

Reader Service 43

Gould offers Tempest superminicomputers

Gould's Federal Systems Division has announced two Tempest superminicomputers. The new computers are Tempest versions of Gould's PowerNode 9000 and Concept/32 computer systems. According to the company, the 9000T and 9700T reach processing speeds of more than 10 million instructions per second and are hardware and software compatible with the midrange 6000T and 6700T products from Gould.

The 9000T is Unix-based, running Gould's UTX/32 and NCSC-rated C-2 secure Unix software. The 9700T runs Gould's proprietary MPX-32 operating system.

Prices for both machines start below \$300,000.

9000T: Reader Service 44
9700T: Reader Service 45

HP color-graphics printer targets PCs

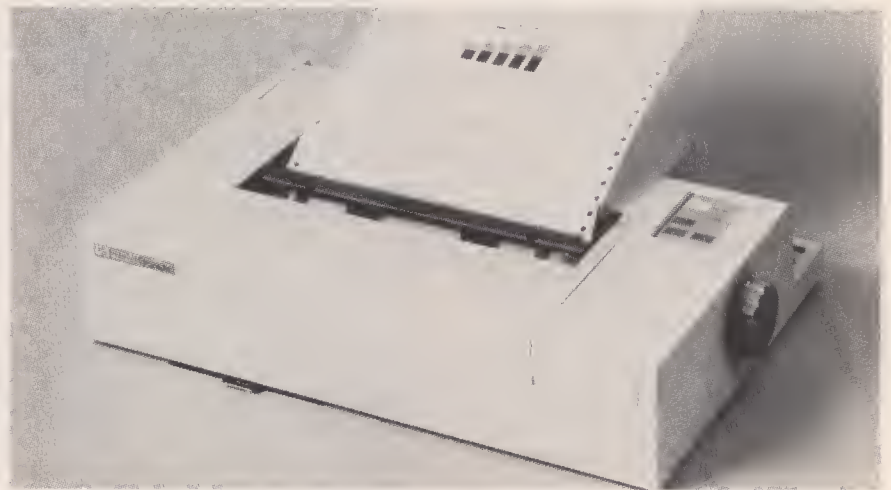
Hewlett-Packard has announced the HP PaintJet color-graphics printer. It produces text and graphics with 180 × 180 dots-per-inch resolution and near letter-quality text at 167 characters per second.

The printer holds four inks—black, yellow, magenta, and cyan. It mixes these colors to produce red, blue, and green. According to the company, the primary colors can be mixed to provide 330 different shades and hues, given the appropriate software. The palette allows selection of up to 16 colors at a time.

Two disposable cartridges, black and color, contain a total of sixty nozzles, inks, and electrical-printing elements.

The printer handles Z-fold or cut-sheet paper and single-sheet transparency film in A or A4 sizes.

Options include RS-232-C/CCITT



The PaintJet thermal color-graphics printer from Hewlett-Packard measures 3.86 × 17.4 × 11.89 inches and weighs 11 pounds.

V.24 serial, Centronics parallel, and HP-IB (IEEE-488) interfaces.

The PaintJet printer costs \$1395.

Cartridges cost \$27.95 for black and \$34.95 for color.

Reader Service 46

Apollo offers Tempest versions of workstations, servers

Apollo Computer has announced Tempest versions of its workstation and server family, including the Domain Series 3000 Personal Workstation, DN580 Turbo graphics workstation, and DSP90 and DFS90 servers.

The new systems—TDN3000, TDN580 Turbo, TDSP90, and TDFS90—were modified to meet requirements of the NACSIM 5100A

specification. The modifications reportedly include physical security additions and structural design changes.

According to the company, the Tempest systems are linked through a fiber ring local area network operating at 12M-bits per second and employing standard 62.5 micron cable with ST connectors.

The Tempest product family runs

Apollo's Domain/IX software, a twin port of Berkeley Unix 4.2 and System V Release 2. Users can run applications in either operating system or both simultaneously from a single workstation. The Tempest versions are software compatible with their commercial counterparts.

Workstations: Reader Service 47
Servers: Reader Service 48

Apollo offers personal super workstations

Apollo Computer has announced the Domain Series 4000 Personal Super Workstation family. According to the company, the new models include a 4-MIPS color workstation for under \$19,000; a 4-MIPS monochrome workstation for under \$14,000; and a 4-MIPS server for under \$13,000.

The Series 4000 Personal Super Workstation family is available in workstation and server configurations. Models include a 25-MHz Motorola MC68020 central processor, 25-MHz MC68881 floating-point coprocessor, virtual cache architecture and real-memory capacity.

According to the company, the new series features the flexible open bus architecture of an IBM PC-AT-compatible expansion bus. The Series 4000 can share files and applications with other Domain workstations, including the Series 500 Turbo workstations.

Reader Service 49



Apollo Computer's Domain Series 4000 Personal Super Workstation family includes 15-inch color (left), 19-inch color (center), and 19-inch monochrome (right) workstations and a server (front). Each system runs directly on Ethernet and Domain Token Ring networks.

Texas Instruments' new Explorers use AI chip

Texas Instruments has announced the Explorer II and Explorer II LX systems.

The Explorer II is based on the company's Explorer Lisp microprocessor, a production version of the 32-bit VLSI processor designed specifically for artificial intelligence applications.

The Explorer II LX system integrates Lisp and Unix environments, combining the Explorer II processor and a 68020-based processor running TI System V.

Explorer software, Release 3.0, provides an environment for developing knowledge-based systems and prototyping of software systems. Networking capabilities now include interfaces to IBM SNA, DECnet, TCP/IP, and NFS. The software comes with all Explorer II systems.

Prices range from \$49,900 to \$99,900.

The base configuration includes the Explorer II processor, 8M bytes of memory, display, keyboard and mouse, 140M-byte Winchester disk, local area network interface, and Explorer system software license. An Explorer II LX system with 32M bytes of memory, three 140M-byte disks, and cartridge tape costs \$99,900.

Reader Service 50



Texas Instruments' Explorer II system provides more than five times the performance of previous Explorer systems, as measured by the Gabriel benchmarks.

Icon micros designed for the classroom

Unisys has announced the Icon Series of microcomputers, designed specifically for educational use.

The Icon Series uses QNX, a proprietary operating system supporting multilingual, curriculum-oriented languages, including Basic, Pascal, C, Fortran, Cobol, APL, and Logo. The company also offers more than 150 courseware packages covering most disciplines for grades Kindergarten through 12, with another 100 under development. According to the company, the Icon Series will also run many MS-DOS-based software.

Each Icon workstation is an Intel 80186 microprocessor with up to one megabyte of memory, high-resolution color graphics, built-in speech synthesis, and a trackball pointing device. They cost \$1895.

Up to 32 workstations can be connected through a hard-disk file server.

The Icon Series is an enhanced version of Meridian Technologies' instructional system, marketed in Canada since 1984.

Reader Service 51

Measure for Symphony collects data

Lotus Development has announced Lotus Measure for Symphony, a PC software package that works from within Symphony to collect data from measurement instruments.

Measure supports three interfaces for data collection: IEEE-488, RS-232-C, and analog-to-digital plug-in boards. It is reputedly compatible with more than 8000 instruments and devices.

The software works with the windows feature in Symphony, allowing users to combine views of data or to compare data from different stages or experimental runs.

Measure for Symphony runs on the IBM PC, PC-XT, and PC-AT, the Hewlett-Packard Vectra PC, and the Compaq Portable. It requires MS-DOS version 2.0 or above and 512K bytes of memory. The company recommends 640K bytes and a hard disk.

Lotus Measure for Symphony costs \$495. The price includes support for RS-232-C and IEEE-488 communication buses, as well as selected data acquisition boards.

Reader Service 52

Lisp runs on Butterfly parallel processors

BBN Advanced Computers has announced Butterfly Lisp, which runs on the Butterfly parallel processor.

Butterfly Lisp supports Scheme and Common Lisp dialects in both interpreted and compiled modes, according to the company, and has been extended to support a multiprocessing environment.

A performance analysis facility allows programmers to analyze the dynamic behavior of Butterfly Lisp programs. It graphically displays the evolution of an algorithm once run on the Butterfly system. A mouse can be used

to point to a thread of control to display the source code that created it.

Runtime information about the state of each processor is also displayed graphically. The programmer can check the status of tasks waiting to be executed, processor utilization, and rate of memory consumption. The language also supports a user interface of windows directly associated with tasks running on the Butterfly system.

Butterfly Lisp is scheduled for release in December 1987. It will cost \$12,000.

Reader Service 53

MIPS system attains 10 MIPS performance

MIPS Computer Systems has announced the M/1000 System, providing sustained performance of 10 million instructions per second, according to the company.

The M/1000 is the third in the MIPS M Series of RISC-based systems. It was designed as an applications server for OEMs building Unix systems. It is available in a base configuration consisting of the R2800 CPU board and 16M bytes of memory in a 12-slot VMEbus cardcage. It costs \$35,900.

Options include up to 64M bytes of additional main memory, a 337M-byte or 689M-byte ESMD disk drive, 1/4-inch tape drive, up to 32 serial ports, and an Ethernet controller.

Software available includes either System V.3 or Berkeley 4.3 versions of

the UMIPS operating system and an optimizing compiler system with a symbolic source-level debugger. Also provided are support for Ethernet, TCP/IP, Sun Microsystem's Network File System, and Sun's PC NFS.

The R2800 CPU board contains the MIPS R2000 CPU, a 32-bit CMOS microprocessor. It features high-speed instruction and data caches, separate buses for I/O and computation, memory interface circuitry, and a coprocessor interface to the MIPS R2010 Floating Point Accelerator. It costs \$10,000 alone.

The M/1000 Upgrade allows users of MIPS' M/500 and M/800 systems to upgrade to the M/1000 for \$10,000.

Reader Service 54

Intel offers its first 80386-based microcomputer

Intel Systems Group offers the System 320, the newest member of Intel's System 300 family and the company's first 80386-based microcomputer system.

According to the company, the 320 is the first 80386-based system configured to support real-time processing using the iRMX operating system. The iRMX operating system is Intel's real-time, multitasking operating system that reportedly provides configurable resources such as interrupt management, and standard device drivers. It also features a new symbolic debugger called Softscope.

The System 320 also supports Intel's Multiserver software. The Multiserver

System 320 links PCs, terminals, minicomputers, mainframes, and applications into a departmental level system.

System 320 configuration options include a base system, the iRMX operating system, languages and tools, Multiserver software, storage devices, channel communications, host communications, Intel's OpenNet networking capabilities, and value added software.

Multiserver System 320 costs \$15,520. The Financial System 320 costs \$12,950. Contact Intel's overseas sales offices for non-US prices.

Reader Service 55

Harris expands H-Series line of superminicomputers

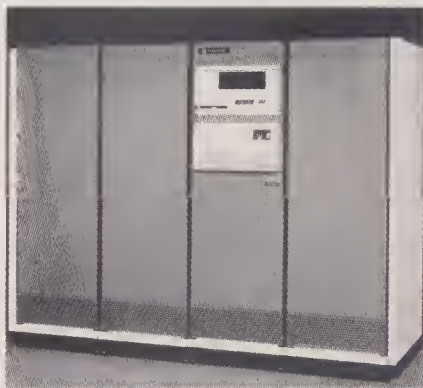
Harris Computer Systems Division has announced four additions to the H-Series line.

The H-1600 reputedly performs at 15 million instructions per second. The triple-processor system can be expanded to a total of 12 processors, with a potential performance of 60 MIPS. It features three 80M-byte per second memory buses, a 40M-byte per second shared memory, 57M-byte per second I/O bandwidth, and in each processor, up to 12M bytes of main memory, three memory subsystems, 6K bytes of cache memory, and up to 288K bytes of bulk cache. The H-1600 is priced from \$795,000.

The dual-processor H-1500 reputedly operates at 10 MIPS, expandable up to 12 processors and 60 MIPS. It features dual 80M-byte per second memory buses, 40M-byte per second shared memory, 38M-byte per second I/O bandwidth, and in each processor, the same features as the H-1600. It costs from \$555,000.

The single-processor H-1100 has up to 12M bytes of main memory including three memory subsystems, 6K bytes of cache memory, up to 288K bytes of bulk cache memory, and 19M-byte per second I/O bandwidth. It costs from \$260,000.

The single-processor H-900 has up to 12K bytes of main memory, 6K bytes of cache memory, and 19M-byte per sec-



The triple-processor Harris H-1600 superminicomputer features 15-MIPS performance.

ond I/O bandwidth, priced from \$240,000.

Each processor also includes integral floating-point hardware, 16 external priority interrupts expandable to 72, line frequency clock, interval timer, and maintenance aid processor. Each system has a combined operator communications console and maintenance aid processor terminal for central system control and remote diagnostics.

The H-1500 and H-1600 use a multi-processor version of the Harris Real-Time Virtual Operating System (RT-



The dual-processor Harris H-1500 superminicomputer features 10-MIPS performance.

VOS). The new operating system, called RT-VOS/MP, reputedly provides single-point control for a multiprocessor complex with up to 12 CPUs. Centralized system loading, configuration, task scheduling, synchronization, and diagnostics are controlled from a single operator console.

RT-VOS/MP supports programs written in Ada, Fortran, C, and Assembler. It is compatible with RT-VOS, VOS, and VUE.

Reader Service 56

DEC adds to VAX AI development tools

Digital Equipment has announced an enhanced version of its VAX Lisp software, Version 2.2, for VMS and Ultrix operating systems. According to the company, developers can now write VAX Lisp applications and deliver them without purchasing separate VAX Lisp runtime licenses for users.

A System Build Utility reputedly allows programmers to build more efficient and smaller programs. Development tools such as the debugger, compiler, and editor can be left out. The resulting single executable image can be shared among multiple users. According to the company, the utility optimizes performance by using operating system management tools to offload image sharing from the Lisp system.

VAX Lisp/VMS supports character input streams for users who dislike the

default provided.

Version 2.2 gives VAX Lisp/Ultrix users access to the Ultrix-32 Graphics Library to take advantage of windowing and other graphics functions.

According to the company, Version 2.2 provides a 20-percent performance increase over Version 2.1.

Digital also offers Nexpert Object, developed by Neuron Data. This graphics-oriented expert system development tool is written in C. A license for the product costs from \$5500 to \$8800.

Nexpert Object and a fully configured VAXstation workstation cost under \$15,000. VAX Lisp Version 2.2 and a VAX system cost under \$7000.

VAX Lisp: Reader Service 57
Nexpert Object: Reader Service 58

EMS support for PS/2

Vericom has announced LIMbo, software that reputedly provides true expanded memory capability for IBM's Personal System/2 Models 50 and 60.

Based on EMS technology developed by Borland International and licensed to Vericom, LIMbo transforms banks of memory on IBM's 80286 Memory Expansion Option board into Lotus/Intel/Microsoft expanded memory, Version 3.2.

According to the company, LIMbo makes use of a page mapping capability on the IBM board and supports up to 8M bytes of expanded memory using multiple boards. The software includes an EMS device driver, RAMdisk, and printer spooler.

LIMbo costs \$49.95.

Reader Service 59

IC Announcements

Company, Model, Function	Comments	R.S. No.
Advanced Micro Devices 80L286 80286 chip	A 12.5-MHz version of the 80286 16-bit microprocessor. Comes in a plastic leaded chip carrier package. Operates at 2.2 watts at 55 degrees C, consumes 30 percent less power than standard-power 80286. Also comes in 8 and 10 MHz versions. Cost: \$100 in 68-pin PLCC (100s).	135
Austek Microsystems A38152 Microcache Cache controller	A 32K-byte cache controller for microprocessors based on Intel's 80386. Features four-way set associativity, 16 and 20 MHz speeds, direct interface to 80386 and four 8K-bit $\times$ 8 SRAMs, 32-bit addressability, on-chip 16-ns tag RAMs, write buffer support, and an 84-lead chip carrier package. Cost: \$58 (10,000s).	136
Bowmar/White Technology C8-M128 SRAM	A 128K $\times$ 8-bit static RAM that operates between -55 degrees C and 200 degrees C. Features a typical access time of 150 ns, an operating current of 5 mA, common data input and output, and a 4.5V to 5.5V power supply. Typical standby current of 2 mA. Comes in a 32-pin DIP. Cost: \$342 (100s).	137
Chips and Technologies 82C437 SharpScan Graphics chip	An EGA-compatible graphics chip with 1128 $\times$ 560-line resolution. Features a pixel multiplexing function that permits a trade-off of colors available (four instead of 16) for higher resolution. Does not require additional software or setup changes to coexist with EGA. Includes video drivers. Cost: \$6.70 (1000s).	138
Emulex ESP chip SCSI processor	A VLSI device implementing the SCSI bus. Replaces existing SCSI interface circuitry. Features dual-ranked command and transfer counter, bus sequences implemented without microprocessor intervention, 16-byte FIFO, and single-ended 48 mA bus transceivers. Comes in a 68-pin PLCC. Cost: \$25 (1000s).	139
Future Domain TMC-900 SCSI interface	An SCSI interface for Intel 8088, 8086, 80286, and 80386 computers. Designed for IBM PC-XT, PC-AT, or 386 compatibles. Interfaces up to seven SCSI peripherals. Software available to run SCSI peripherals under DOS, Xenix, or Novell. Contact the company for pricing.	140
Gain Electronics GFL1700, GFL3500, FGL6000 GaAs gate array family	Compatible with ECL, TTL, and CMOS logic. Feature 100-ps unloaded gate delay. GFL1700 has 1672 equivalent gates, 92 I/O buffers, and 1W maximum chip power. GFL3500 has 3456 gates, 140 I/O buffers, and 2W maximum power. FGL6000 has 5776 gates, 204 I/O buffers, and 3W maximum power. No prices given.	141
Integrated Device Technology IDT7134 SRAM	A CMOS static RAM, organized 4K $\times$ 8-bit. Operates at 35 ns. Separate control, address, and I/O pins for its two independent ports. Features independent, asynchronous access for read or write functions. Zero wait state operation. Commercial or military versions, in a 48-pin sidebrazed DIP or 52-pin LCC. Cost: \$60 (100s) for commercial plastic DIP.	142
International CMOS Technology 27CX641-45, 27CX641-55 EPROMs	Members of a 24-pin family of EPROMs with speed ranges of 35 ns, 45 ns, and 55 ns. Available in 64K (8K $\times$ 8) and 32K (4K $\times$ 8) densities in 300 mil or 600 mil windowed packages. 27CX641 is an 8K $\times$ 8 EPROM in a 600-mil package. Cost: \$49.80 for the 27CX641-45 (100s); \$41.50 for the 27CX641-55 (100s). Contact the company for further pricing.	143
Linear Technology LTC1091 Data acquisition	A monolithic data acquisition system with a 20 microsecond conversion time. Combines a 10-bit A/D converter, a two-channel multiplexer, a built-in sample-and-hold, and a serial I/O interface, all under software control. Comes in an 8-pin ceramic or plastic DIP. Cost: \$10.95 (100s) for commercial grade.	144
LSI Logic L642xx Image processing	A four-chip family of image processing ICs that work in real time. Fabricated in 1.5-micron HCMOS, operate at 20 million cycles per second over commercial temperature and voltage ranges. Also available in military versions. L64210/L64211: variable length video shift registers. L64220: rank value filter. L64230: binary filter and template matcher. L64240: multibit filter. No prices given.	145

Microsystem Announcements

Company, Model, Function	Comments	R.S. No.
Arcom Control Systems SEERAM Memory board	A nonvolatile memory board for the STEbus. Can be fitted with up to 256K of battery-backed CMOS RAM, EPROM, or EEPROM. Comes with eight 28-pin memory sockets and on-board Ni-Cad battery with trickle charger circuitry. Accepts 8K or 32K devices. Cost: £135.	120
Arcom Control Systems DIO-5 Digital I/O board	Provides digital I/O lines for the ARCBus via a standard SCB ribbon connector interface. Uses two 8255s to provide 40 lines, 32 fully buffered, 16 that can be disabled at power-up. Software programmable direction and enable for each of four groups of eight lines. Cost: under £200.	121
Bubbl-Tec BDJ-1, BDJ-2 Bubble-memory systems	The BDJ-1 Bubbl-Dek system is a full-height replacement for a 5¼-inch floppy disk drive. It provides two front-panel slots for plug-in Bubbl-Pac cartridges. The BDJ-2 is a half-height unit that accommodates one Bubbl-Pac. Cost: \$787 for the BDJ-2, \$1291 for the BDJ-1 (quantity ten).	122
Computer Peripherals Hook-Up Internal modem	A half-slot plug-in modem board operating at 1200 or 0-300 baud. Can be plugged into an IBM PC, PC-XT, PC-AT, or compatible. Compatible with Hayes Smartmodem 1200 B and Bell 103 A/J and 212A. Optional communications port, compatible with RS-232-C serial protocol. Bundled with PC Talk software. Cost: under \$200.	123
Franklin Telecommuni- cations MacTwenty External hard disk	A 20M-byte hard disk drive with an SCSI connector for the Apple Macintosh SE and Macintosh Plus. Has four AC power outlets for plug-in of other peripherals. Can accommodate six additional SCSI devices. Features self-diagnostics, microstepping, noise filter, and surge suppression. Cost: \$850.	124
Integrated Solutions VME-SCSI/U SCSI host adapter board	A universal SCSI host adapter board that supports any SCSI target device conforming to ANSI X3.131 and complying with the Common Command Set. Occupies one slot on the VME backplane. Maximum transfer rate between target SCSI device and VMEbus of approximately 1.5M-bytes per second. Includes a 16-bit processor, 16K-byte data buffer memory, and self-test diagnostics. Cost: \$1795.	125
Maxtor LXT-170 3½-inch Winchester drive	First in a family of 3½-inch Winchester drives. Stores 170M bytes with average access times of less than 20 ms. Comes with either SCSI or ESDI. Features dedicated track following, closed-loop servo system, and embedded spindle motor. Achieves a recording density of 23,897 bits per inch. Cost: around \$1000 for OEMs.	126
Megahertz EasyTalk EMS Memory/modem card	A multifunction card for Toshiba's T1100 Plus. Features 1M byte of expanded memory and an internal, Hayes-compatible 1200/300-baud modem. Comes bundled with Crosstalk communications software. Memory conforms to L1M EMS. Optional ultra-low power memory to extend battery life (\$200). Cost: \$899.95.	127
Micro Solutions Z80 card CP/M card	A Z80 card that allows running of CP/M programs on an IBM PC or compatible. Runs at 8 MHz with no wait states. Includes 64K of on-board memory and fits in one half-size expansion slot. Comes with UniDOS, a CP/M emulator. Cost: \$195.	128
Mitsubishi Electronics America MF353C, MF355C 3½-inch drives	Double-sided 3½-inch flexible disk drives that occupy 80 percent the space of half-height drives. The MF353C has 1M-byte of unformatted memory capacity; the MF355C supports both 1M-byte and 2M-byte capacities (read/write). The MF355C is downwardly compatible with the MF353C. Cost: OEMs and industrial distributors, contact Mitsubishi.	129
Plessey Microsystems PME 98-XX Analog I/O boards	PME 98-07 has resistor-selectable gain parameters and 32 single-ended or 16 differential input channels. PME 98-06 has 64 single-ended or 32 differential outputs with 12-bit resolution, software-selectable gain parameters, and optional digital I/O module. PME 98-52 offers eight-channel analog input with 12-bit resolution and three-microsecond throughput. PME 98-04 has 16 analog output channels, 12-bit resolution, and four user-selectable output ranges. PME 98-54 features eight output channels, programmable conversion rates, and selectable output ranges. No prices given.	130



IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN

ICCAD-87

A CONFERENCE FOR THE
EE CAD PROFESSIONAL

November 9-12, 1987

Convention Center, Santa Clara, CA

PROGRAM HIGHLIGHTS

Sponsored by
IEEE COMPUTER SOCIETY
IEEE CIRCUITS AND SYSTEMS SOCIETY
In Cooperation with
IEEE ELECTRON DEVICES SOCIETY
ACM SPECIAL INTEREST GROUP
ON DESIGN AUTOMATION

THE CONFERENCE

The 1987 IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN will be held November 9 - 12, 1987, in Santa Clara, CA. The conference concentrates on CAD for electronic circuit design, and features 1 day of tutorials and workshops, and 3 days of technical papers from an internationally recognized group of CAD experts.

TECHNICAL SESSIONS

The thirty-five technical sessions at ICCAD-87 fall into four general categories: systems, simulation, test, and layout. One hundred nineteen papers will be presented in three parallel sessions. A partial list of session topics is shown below.

- | | |
|-------------------------------------|--|
| • Integrated Circuit Layout | • Timing Analysis |
| • Block Placements | • Finite State Machine Design |
| • Logic Synthesis | • IC Layout Compaction |
| • Design for Testability | • Hardware Simulation Accelerators |
| • Simulation on Processor Arrays | • Hardware for Floorplanning and Placement |
| • Automatic Test Pattern Generation | • Analog and DSP Synthesis |
| • Yield & Reliability Enhancement | • Device and Technology Modeling |
| • Module Generation | • Fault Simulation |

PANEL SESSIONS

The evening of Tuesday, Nov. 10, 1987 will feature two outstanding panel sessions.

- | | |
|-----------------------------------|-----------------------|
| • Mixed Analog/Digital Simulation | • Simulated Annealing |
|-----------------------------------|-----------------------|

TUTORIALS AND WORKSHOPS

On Monday, Nov. 9, 1987, ICCAD-87 will sponsor five all-day tutorials and workshops. The popular tutorial sessions have limited space requiring early registration.

- | | |
|--------------------------------------|---------------------------------------|
| • Introduction to Circuit Simulation | • Logic Synthesis |
| • VHDL | • AI Applications for Test Generation |
| • EDIF Workshop | |

FOR FURTHER INFORMATION

Contact M.P.Associates, 7366 Old Mill Trail, Suite 101, Boulder, CO 80301. Telephone: (303) 530-4562, or FAX your request (303) 530-4334.



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

acm Association for Computing Machinery



THE COMPUTER SOCIETY
OF THE IEEE

HOTEL REGISTRATION FORM
5TH INTERNATIONAL CONFERENCE ON
COMPUTER-AIDED DESIGN
November 9 - 12, 1987

Room reservations must be made directly with the hotel of your choice. A block of rooms has been reserved at the Doubletree and at the Santa Clara Marriott. The Doubletree is located beside and attached to the Santa Clara Convention Center and the Marriott is about four blocks away.

ROOM RATES:

Doubletree	\$85.00 single	\$95.00 double
Marriott	\$87.00 single	\$97.00 double

Reservations received after October 26, 1987 are subject to availability.

Be sure to specify that you are attending ICCAD when calling the hotel for reservations to assure that you receive the conference rate, or complete the form below and send to:

Doubletree Hotel	Santa Clara Marriott
5101 Great America Pkwy.	2700 Mission College Blvd.
Santa Clara, CA 95054	Santa Clara, CA 95054
(800) 528-0444	(408) 988-1500

PLEASE NOTE: Hotel reservation forms sent to conference registration will be discarded.

Name _____ Phone _____

Company _____ Mail Stop _____

Address _____

City _____ State _____ Zip _____

Arrival Date _____ Time _____ AM

Departure Date _____ Time _____ PM

Your reservation will be confirmed by the hotel. A first night's deposit is required to confirm your reservation for arrival after 6:00 p.m..

_____ Visa _____ American Express _____ Mastercard

Number _____ Expiration Date _____

ADVANCE CONFERENCE REGISTRATION FORM
5TH INTERNATIONAL CONFERENCE ON
COMPUTER-AIDED DESIGN
November 9 - 12, 1987

REGISTRATION DESK

Advance registrants may pick up the conference materials and others may register during the conference at the registration desk located in the Doubletree Hotel on the Second Floor (Ballroom area).

The registration desk will be open:

Sunday, November 8 2:00 p.m. to 6:00 p.m.
Monday, November 9 7:30 a.m. to 9:00 a.m.
Monday, November 9 9:00 a.m. to 9:00 p.m.
Tuesday, November 10 8:00 a.m. to 4:00 p.m.
Wednesday, November 11 8:00 a.m. to 4:00 p.m.
Thursday, November 12 8:00 a.m. to 2:00 p.m.

*This time slot is only for those who also have registered for tutorials.

ADVANCE CONFERENCE REGISTRATION FORM
5TH INTERNATIONAL CONFERENCE ON
COMPUTER-AIDED DESIGN
November 9 - 12, 1987

REGISTRATION FEES

Conference registration fee includes breakfast and lunch Tuesday through Thursday, cocktail parties on Monday and Tuesday evenings, banquet dinner Wednesday evening, and conference Digest of Technical Papers.

Before Oct. 9 1987 After Oct. 9 1987

IEEE/ACM Members	\$140	\$180
Non-members	\$180	\$230
Students*	\$ 30	\$ 30

(*Students must be members of IEEE or ACM. Student registration covers only attendance at technical sessions and digest.)

Conference Plus One Tutorial**

IEEE/ACM Members	\$290	\$375
Non-members	\$375	\$480

Workshop Fees

EDIF	\$ 75	\$ 75
------	-------	-------

TUTORIALS: (Please indicate first and second choices.)

_____ ** Introduction to Circuit Simulation
_____ ** Artificial Intelligence Applications for Test Generation
_____ ** Logic Synthesis
_____ ** VHDL

PLEASE NOTE THAT ONLY CONFERENCE REGISTRANTS CAN ATTEND TUTORIALS. To be guaranteed a tutorial, please select more than one. You will be refunded if your choice(s) cannot be filled. Also, the EDIF Workshop runs the same time as the tutorials. You can only attend one or the other. (Breakfast and lunch are included in the tutorial fee.)

WORKSHOP:

_____ **EDIF Workshop** (If interested in attending EDIF Workshop only -- no technical sessions -- register at conference.)

Advance registration for tutorials and workshops is advised since attendance is limited, and must be received before October 9, 1987. At-site conference payments can be made by check or cash - **NO CREDIT CARDS ACCEPTED.** No refunds unless cancellation is received before October 15, 1987.

Name _____ Phone _____

Company Affiliation _____

Address _____

City _____ State _____ Zip _____

Country _____ IEEE/ACM Membership No. _____

_____ Speaker _____ Session Chairperson

Conference/Tutorial/Workshop fees enclosed:

\$ _____ IEEE/ACM Member \$ _____ Non-member \$ _____ Student \$ _____ EDIF

Total fees: \$ _____

PLEASE MAIL REGISTRATION FORM & CHECKS MADE PAYABLE TO:

MP ASSOCIATES, INC.

7366 Old Mill Trail - Suite 101, Boulder, CO 80301.
Phone: (303) 530-4562

CONFERENCES

Editor: Edmund L. Gallizzi, Computer Science Dept., Eckerd College, St. Petersburg, FL 33733; (813) 867-1166 x272

Broad FJCC agenda to delve into software, other aspects of industry's future

Because of the important role software is expected to play in the next decade, a unique and extensive software track will provide one of the key focuses of the broad program slated for the Fall Joint Computer Conference October 25-29 at the Infomart in Dallas.

According to Program Co-chair Raymond Yeh, the overall technical program has been designed to achieve three main objectives—high quality, general interest, and an international flavor.

A session on alternative paradigms that challenge current software development methods will launch the software track, one of five major tracks on the agenda.

One of the track's highlights will be the presentation of two closely related sessions, one entitled "Software Problems of Large Organizations," and the other "Solutions to Large Software Development Problems."

The invited participants of these back-to-back sessions include Elaine Bond, vice president of Chase Manhat-

tan Bank; Susan Hoben, director of MIS for Travelers Insurance; Mike Schumacker, senior vice president of Control Data Corporation; Terry Straeter, senior vice president of General Dynamics; Denji Tajima, managing director of Hitachi Software Company; Bill Riddle, executive vice president of the new Software Productivity Consortium; and Larry Druffle, director of the Software Engineering Institute at Carnegie Mellon University.

Sessions on software design synthesis in application domains will look at dependable air traffic control software for the FAA, strategic computing, and artificial intelligence.

Besides Software, other tracks on the agenda include International, Super Computer/Parallel Computations, Database/Computer-Aided Design, and AI/Expert Systems and Theory.

In "Software Development in Japan," an invited paper session within the International track, a paper entitled "The Influence of Culture and Social Aspects on the Japanese Software

Development" by Tajima and his coauthors will provide excellent insight into the Japanese software industry.

Other session members will include Yoshihiro Matsumoto, manager of the Toshiba Software Development Factory where, according to Yeh, software production is 10 times the US average, and Kiichi Fujino, vice president of NEC in charge of software research and development. This session, coupled with papers on a European software factory project, China's new high-tech program, and Brazil's national software factory, will provide an indication of the fierce global software competition that experts predict will occur in the next decade.

The Parallel Computations track will include three sessions featuring various dataflow architectures being researched in Japan. Likewise, three sessions are included in the Database/Computer-Aided Design track, with heavy emphasis on the application of AI techniques to CAD.

The AI/Expert System track will feature five sessions, highlighted by an AI/software engineering parallel. Two sessions are included in the Theory track, with primary concentration on algorithm design.

Other sessions will feature such topics as computer security, fault-tolerant computation, computer communications, image recognition/signal processing and visual languages.

According to Conference Chair Stephen Szygenda, many of the presentations will discuss international impacts on the computer industry and should be of considerable interest to technology management.

"While there are many excellent specific technical presentations to be given, the global aspects of our industry will be emphasized at this conference, since it is the annual conference of the ACM and IEEE Computer Society," Szygenda said.

The opportunity to visit Infomart's permanent facilities will add another dimension to this year's conference, he said.

Discounts are available if you register in advance on or before September 25 to attend FJCC. For complete information, see pages 12-15 of this issue of *Computer*.

AFIPS Board votes to continue NCC series

Despite the urgings of the Computer Society Board of Governors, the American Federation of Information Processing Societies Board has voted to continue the ailing National Computer Conference series.

The AFIPS panel met in an assembly session July 22 and, taking a straw vote, decided 9-8 to recommend that the policy-making National Computer Conference Board proceed with plans to stage future NCCs.

Supporting the Computer Society's position that NCC be discontinued were representatives from the Association for Computing Machinery and the Society for Information Display. The Computer Society and ACM are co-owners of NCC along with the Society for Computer Simulation, the Data Processing Management Association, and AFIPS, a federation which includes seven other professional societies.

Subsequently, the NCC board voted 5-3 on July 30 to approve an agreement

through which ISASI, a for-profit subsidiary of the Instrument Society of America, will assume the management of future NCCs. The Computer Society voted against retaining ISASI.

In recent years, NCC has suffered declines in attendance and in the volume of exhibitor participation.

Computer Society President Roy Russo has indicated that, since the CS Board of Governors passed a resolution in June enabling withdrawal from the NCC agreement (see August *Computer*, p. 110), the society's responsibility will end with participation in NCC 88, slated May 31 to June 3 at the Los Angeles Convention Center.

The CS board passed its resolution after several months of discussion and study, concluding that NCC, once a viable part of the professional community's calendar, had outlived its usefulness. At its peak, NCC drew an estimated 100,000 persons but, by 1987, attendance had slipped to about 14,000.

ACM to mark 40th anniversary at FJCC 87

The Association for Computing Machinery will observe its 40th anniversary at the Computer Society/ACM Fall Joint Computer Conference next month in Dallas.

During a 2 to 4 p.m. history session October 27 in the Infomart conference facility, panelists Adele Goldberg, David Brandin, Peter Denning, Anthony Ralston, Paul Armer, and Alan Perlis will present recollections and comments on the organization's first four decades.

Walter Carlson, who served as ACM president in 1970-72 and will chair the Dallas session, said the panelists will discuss key turning points and personalities.

ACM "has grown up with the industry," said Anita Cochran, chair of the organization's 40th anniversary committee.

"We were founded less than a year after the announcement of Eniac, the world's first electronic computer, and John Mauchly, co-inventor of the Eniac, was one of our founders," she said.

With more than 75,000 members, 31 special interest groups and 500 chapters around the world, ACM has "good reason to celebrate," she added.

Two from CMU keynote FTCS 87

Larry Druffel and Raj Reddy of host Carnegie Mellon University delivered the keynote remarks at the 17th International Fault-Tolerant Computing Symposium in Pittsburgh July 6-8.

Druffel, director of CMU's Software Engineering Institute, discussed the difficulties involved in the design, development, and maintenance of software for complex computing systems. He said these problems, including software reliability issues, are challenging but not insurmountable.

Reddy, director of the Robotics Institute, summarized a number of ongoing research projects in the areas of automated systems and robotics, and described potential synergisms involving research into artificial intelligence and fault-tolerant computing.

Steve Director, head of the Dept. of Electrical and Computer Engineering, announced the establishment of a new research center on dependable computing systems on the campus.

CALENDAR

September 1987

Workshop on the Role of Optical Sensors in Power Systems Voltage and Current Measurements (NBS), September 16-18, Gaithersburg, Maryland. Contact Raymond S. Turgel, B162 Metrology Building, National Bureau of Standards, Gaithersburg, MD 20899; (301) 975-2420.

Historical Symposium on Science and Engineering in the Public Interest (NBS), September 17-18, Gaithersburg, Maryland. Contact Sara R. Torrence, A902 Administration Building, National Bureau of Standards, Gaithersburg, MD 20899; (301) 975-2774.

33rd IEEE Holm Conference on Electrical Contacts, September 21-23, Chicago. Contact IEEE Holm Conference Registrar, 345 E. 47th St., New York, NY 10017-2394; (212) 705-7405.

 **CSM-87, Conference on Software Maintenance (AWC, DPMA, NBS, SMA), September 21-24,** Austin, Texas. Contact Roger Martin, National Bureau of Standards, Bldg. 225, Rm. B266, Gaithersburg, MD 20899; (301) 921-3545 or Computer Society of the IEEE, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903; (202) 371-0101.

10th National Computer Security Conference, September 21-24, Baltimore, Maryland. Contact Irene Isaac, B266 Technology Building, National Bureau of Standards, Gaithersburg, MD 20899; (301) 975-3360.

 **ICDCS-7, Seventh International Conference on Distributed Computing Systems, September 21-25,** Berlin. Contact R. Popescu-Zeletin, Hahn-Meitner-Institut, Berlin, Glienicke Strabe 100, D-1000, Berlin 39, West Germany; phone 49 (30) 80-09-2594.

Northcon 87 (IEEE), September 22-24, Portland, Oregon. Contact Dale Litherland, Electronic Conventions Management, 8110 Airport Blvd., Los Angeles, CA 90045; (213) 772-2965.

ACM SIGUCCS User Services Conference XV, September 27-30, Kansas City, Missouri. Contact M. Lloyd Edwards, Emporia State University, 12th and Commercial, Emporia, KS 66801; (316) 343-1200.

Vibrations Conference and 1987 Design Automation Conference (ASME), September 27-30, Boston. Contact S.S. Rao, School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907; (317) 494-5699 or ASME, 345 E. 47th St., New York, NY 10017; (212) 705-7057.

 **International Conference on Software Engineering for Real-Time Systems, September 28-30,** Cirencester, England. Contact R. Larry, Institute of Electronic and Radio Engineers, 99 Gower St., London WC1E 6AZ, England UK.

 **Oceans 87, September 28-October 1,** Halifax, Nova Scotia, Canada. Contact Alan R. Longhurst, Bedford Institute of Oceanography, PO Box 1006, Dartmouth, Nova Scotia B2Y 4A2, Canada.

Symposium on Infocom Technologies in Medical Applications: Opportunities and Responsibilities (IEEE, AMA, FDA, HIMA, NEMA), September 29-30, Rockville, Maryland. Contact Judith Prewitt, 19 Hidden Hollow Ter., Holmdel, NJ 07733; (201) 888-0882; or Roger Schneider, FDA/NCDRN, 12720 Twinbrook Pkwy, Rockville, MD 20857; (301) 443-5860.

Ninth Annual Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD Association, IIT Research Institute), September 29-October 1, Orlando, Florida. Contact EOS/ESD Symposium, Box 14, Gillette, NJ 07933; (201) 522-4770.

Fifth International Symposium on Data Analysis and Informatics (INRIA), September 29-October 2, Versailles, France. Contact INRIA, Service des Relations Exterieures, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France; phone 33 (1) 39-63-56-00.

25th Annual Allerton Conference on Communication, Control, and Computing, September 30-October 2, Monticello, Illinois. Contact P.R. Kumar or Michael C. Loui, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, 1101 W. Springfield Ave., Urbana, IL 61801.

October 1987

1987 Far East Computer and Software Technology Tour, October 3-18, Japan, Taiwan, and South Korea. Contact Commerce Tours International, 870 Market St., Suite 920, San Francisco, CA 94102; (415) 433-3072, or Gordon MacFarland, JCT International Ltd., Hanover Court, 5 Hanover Square, London W1R 9 HE, England, UK; phone 44 (01) 499-6827.

Object-Oriented Programming Systems, Languages and Applications Conference (ACM), October 4-8, Orlando, Florida.

Contact Donna Baglio, ACM, 11 W. 42nd St., New York, NY 10036; (212) 869-7440.

ASIS 87, 50th American Society for Information Science, October 4-8, Boston. Contact ASIS, 1424 16th St. NW, Washington, DC 20036; (202) 462-1000.

 **Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, October 5-7**, Seattle. Contact Steve Tanimoto, Dept. of Computer Science, FR-35, University of Washington, Seattle, WA 98195; (206) 543-1695.

 **10th Data Communications Symposium (ACM), October 5-7**, Napa, California. Contact Bart Stuck, Probe Research, PO Box 590, Morristown, NJ 07960; (201) 285-1500.

 **12th Conference on Local Computer Networks, October 5-7**, Minneapolis, Minnesota. Contact Stephane Johnson, Start, Inc., 10301 Toledo Ave. South, Bloomington, MN 55437; (612) 831-2122.

AAAI Workshop on Spatial Reasoning and Multisensor Fusion, October 5-7, St. Charles, Illinois. Contact Su-shing Chen, Dept. of Computer Science, University of North Carolina, Charlotte, NC 28223; (704) 547-4880; Tosiya L. Kunii, c/o Business Center for Academic Societies Japan, Yamazaki Bldg. 4F, 2-40-14, Hongo, Bunkyo-ku, Tokyo 113, Japan; phone 81 (3) 817-5831; or Albert K. Hawkes, Sargent & Lundy, Engineering Consultants, 55 E. Monroe, Chicago, IL 60603; (312) 269-3640.

IFIP Conference on Distributed Processing, October 5-7, Amsterdam. Contact M.H. Barton, Electrical Engineering, University of Bristol, BS8 1TR, England, UK.

 **ASPLOS-II, Second International Conference on Architectural Support for Programming Languages and Operating Systems (ACM), October 5-8**, Palo Alto, California. Contact Martin Freeman, Signetics Corp., 811 E. Arques Ave., Sunnyvale, CA 94088-3409; (408) 991-3591.

 **ICCD-87, IEEE International Conference on Computer Design: VLSI in Computers and Processors, October 5-8**, Rye Brook, New York. Contact Prathima Agrawal, AT&T Bell Laboratories, 600 Mountain Ave., Rm. 3D-480, Murray Hill, NJ 07974; (201) 582-6943.

IWDM-87, Fifth International Workshop on Database Machines (ACM, IPSJ), October 5-8, Karuizawa, Nagano Prefecture, Japan. Contact M. Kitsuregawa, Institute of Industrial Science, University of Tokyo, 7-22-1, Roppongi, Minato-ku, Tokyo 106, Japan; phone 81 (03) 402-6231.

 **Compsac 87, 11th Annual Computer Software & Applications Conference (IPSJ), October 5-9**, Tokyo. Contact Stephen S. Yau, Northwestern University, Dept. of Electrical Engineering and Computer Science, Evanston, IL 60201; (312) 491-3641.

Manufacturing Automation Protocol Workshop (SME), October 7, Phoenix, Arizona. Contact Debbie Clark, Technical Activities Dept., Society of Manufacturing Engineers, 1 SME Dr., PO Box 930, Dearborn, MI 48121; (313) 271-1080.

International Workshop on AI Applications to CAD Systems for Electronics (IEEE), October 8-10, Munich, West Germany. Contact Werner Sammer, Siemens AG, Corporate Research and Technology, Otto-Hahn-Ring 6, 8000 Munich 83, West Germany; phone 49 (89) 636-3348 or Alfred C. Weaver, Lockheed EMSCO, MS B-08, 2400 NASA Rd. One, Houston, TX 77058; (713) 333-6792.

 **Seventh Annual Symposium on Small Computers in the Arts, October 8-11**, Philadelphia. Contact Richard Moberg, 338 S. Quince St., Philadelphia, PA 19107; (215) 834-1511.

 **Second Workshop on Large-Grained Parallelism, October 11-14**, Somerset, Pennsylvania. Contact Maurice Herlihy, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213; (412) 268-2584.

Astronomy from Large Databases: Scientific Objectives and Methodological Approaches, October 12-14, Garching-bei-Munchen, West Germany. Contact F. Murtagh, ST-ECF/ESO, Karl Schwarzschild-Str. 2, D-8046 Garching-bei-Munchen, West Germany.

 **FOCS-87, October 12-14**, Los Angeles. Contact Ashok Chandra, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598; (914) 945-1752.

Ultratech 87(SME), October 14-16, Long Beach, California. Contact SME, 1 SME Dr., Dearborn, MI 48121; (313) 271-0023.

IEEE Fifth Biennial Careers Conference, October 14-16, San Diego. Contact William R. Anderson, IEEE, 1111 19th St., NW, Suite 608, Washington, DC 20036-3690; (202) 785-0017.

International Symposium on Methodologies for Intelligent Systems (ACM), October 14-17, Charlotte, North Carolina. Contact Zbigniew W. Ras, Dept. of Computer Science, University of North Carolina, Charlotte, NC 28223; (704) 547-4567.

 **Conference on Data and Knowledge Systems for Manufacturing and Engineering (ACM), October 19-20**, East Hartford, Connecticut. Contact Fred Maryanski, University of Connecticut, Computer Science and Engineering Dept., U-155, Storrs, CT 06268; (203) 486-2584.

Conference on Software for Factory Automation(IFIP), October 19-21, Tokyo. Contact IFIP, 3 rue de Marche, CH-1204, Geneva, Switzerland.

 **Third Expert Systems in Government Conference, October 19-23**, Washington, DC. Contact Peter Bonasso, Mitre Washington AI Center, 7725 Colshire Blvd., MS W952, McLean, VA 22102; (703) 883-6908.

Workshop on Planning a Manufacturing Information System (SME), October 21, San Mateo, California. Contact Debbie Clark, Technical Activities Dept., Society of Manufacturing Engineers, 1 SME Dr., PO Box 930, Dearborn, MI 48121; (313) 271-1080.

Conference on Attacking Legal Problems of the Computer Age (IFIP), October 21-23, Santa Monica, California. Contact IFIP, 3 rue de Marche, CH-1204, Geneva, Switzerland.

 **FJCC-87, Fall Joint Computer Conference (ACM), October 25-29**, Dallas. Contact Debra Anthony, Texas Instruments, 6500 Chase Oaks Blvd., PO Box 86905, MS 8419, Plano, TX 75086; (214) 575-2151.

Gomac 87, Government Microcircuit Applications Conference, October 27-29, Orlando, Florida. Contact Frank J. Rehm, RADC/OC Griffiss Air Force Base, NY 13441-5700; (315) 330-7781.

CCDC-87, Computer Communications for Developing Countries, October 27-30, New Delhi, India. Contact S. Ramani, National Center for Software Technology, Gulmohar Cross Road No. 9, Juhu, Bombay 400049, India; phone 629574 or 629606.

 **Applied Imagery Pattern Recognition, October 28-30**, Washington, DC. Contact Jane Harmon, 403 Argus Pl., Sterling, VA 22107; (703) 351-2708.

AI/East 87, Artificial Intelligence and Advanced Computer Technology Conference/Exhibition, October 28-30, Atlantic City. Contact Tower Conference Management Co., 331 W. Wesley St., Wheaton, IL 60187.

November 1987

 **Workshop on Workstation Operating Systems, November 5-6**, Cambridge, Massachusetts. Contact Luis-Felipe Cabrera, 6572 Northridge Dr., San Jose, CA 95120; (408) 927-1838.

 **Sixth International Conference on Entity-Relationship Approach (ACM), November 9-11**, New York City. Contact Martin Modell, 1 Mohawk Lane, Commack, NY 11752.

 **ICCAD-87, IEEE International Conference on Computer-Aided Design, November 9-12**, Santa Clara, California. Contact Basant Chawla, AT&T Bell Laboratories, 1247 S. Cedar Crest Blvd., Allentown, PA 18103; (215) 770-3484.

CALL FOR PAPERS

Call for papers and referees for Computer

Computer is seeking articles on **multi-valued logic** for inclusion in the special April 1988 issue.

Suggested topics include:

- device technology
- fuzzy logic
- optical computing
- algebraic aspects
- logic design
- applications

Submit 10 copies of each manuscript by **September 15, 1987**, to Jon T. Butler, Northwestern University, Dept. of Electrical Engineering and Computer Science, Evanston, IL 60201; (312) 491-5628.

Persons interested in serving as referees for this issue are asked to contact Butler and to send him a list of their technical interests.

Referees are also sought for special issues that will cover the following topical areas:

- laser communication technology and systems
- computer-integrated manufacturing
- national computer policies—the computer industry in an international context
- innovative printer technology
- storage technology
- concurrent systems: the hardware, production, maintenance, and software issues

- emerging technologies for computer component, hardware, and software design and production
- high-performance CAD/CAM, engineering/scientific, programming workstations
- computers in automobiles
- computers in the entertainment and advertising industries
- real-time systems
- Videotext—has the vision failed?

Persons interested in serving as referees are asked to circle Reader Service Number 195 on the Reader Service Card at the back of this magazine and to send the card to the Reader Service Inquiries Dept.

 **Computer Society of the IEEE Technical Committee on Computer Education:** Contributions up to five typewritten pages are welcomed for the TCCE newsletter, a forum for the exchange of ideas among persons interested in computer education or computers in education. Direct news items, short articles, and any correspondence to Helen Hays, Dept. of Computer Science, Southeast Missouri State University, Cape Girardeau, MO 63701; (314) 651-2244.

International Journal of Computer Applications in Technology begins publication in Spring 1988. Papers, along with 100 to 150-word abstracts, are sought for this UNESCO-supported journal. Send three copies of papers to M. A. Dorgham, The Open University, Milton Keynes, MK7 6AA, England, UK; phone (44) 09-08-653945.

 **Computer Networking Symposium:** April 11-13, 1988, Arlington, Virginia. Submit four copies of the complete paper or of a 1000-word extended abstract by **September 15, 1987**, to Jeffrey Jaffe, IBM Research Center, PO Box 704, Yorktown Heights, NY 10598.

IEEE International Conference on Robotics and Automation: April 25-29, 1988, Philadelphia. Papers 15 to 20 pages long are sought, as are papers five to seven pages

long. Submit papers in either category by **September 15, 1987**, to Robert B. Kelley, ECSE Dept., Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

 **Fourth IEEE Conference on Artificial Intelligence Applications:** March 14-18, 1988, San Diego, California. Full-length papers (5000 words maximum) and poster-session abstracts (1000 words maximum) that describe ongoing research are sought. Submit four copies of the full-length paper (include a 100-word abstract) by **September 18, 1987**, to Elaine Kant or Dennis O'Neill, Schlumberger-Doll Research, Old Quarry Rd., Ridgefield, CT 06877-4108. Submit four copies of poster-session abstracts to Elaine Kant or Dennis O'Neill by **November 25, 1987**. Demonstration proposals and panel session proposals are also sought and are due by **November 25, 1987**.

 **COIS-88, Conference on Office Information Systems:** March 23-25, 1988, Palo Alto, California. Send five copies of papers by **September 21, 1987**, to Robert B. Allen, 2A-367, Bell Communications Research, Morristown, NJ 07960; (201) 829-4315.

CHI-88, Conference on Human Factors in Computing Systems (ACM): May 15-19,

1988, Washington, DC. Send five copies of paper (3000 words maximum) by **September 21, 1987**, to Sylvia Sheppard, Computer Technology Associates, Inc., 14900 Sweitzer Lane, Suite 201, Laurel, MD 20707; (301) 369-2422. Proposals for panel sessions, interactive poster sessions, demonstrations of experimental or commercial user interfaces, video presentations, meetings of special-interest groups, and presentations by PhD students on their dissertations are also sought.

Sixth National Conference on Ada Technology: March 14-17, 1988, Washington, DC. Abstracts of 300-500 words are sought by **September 30, 1987**, on Ada in the life cycle; send to Al Rodriguez, US Army Communications-Electronics Command, AMSEL-RD-LC-ASST-1A, Fort Monmouth, NJ 07703; (201) 532-4725.

Fourth International Conference on Pattern Recognition (BPRA, IAPR): March 28-30, 1988, Cambridge, England. Submit three copies of the complete paper (4000 words maximum) by **September 30, 1987**, to J. Kittler, Dept. of Electronic and Electrical Engineering, University of Surrey, Guildford GU2 5XH, England, UK.

 **IEEE Software:** Articles on compiler environments and technology transfer methods are sought for the May 1988 issue. Contact Ted Lewis, editor-in-chief, *IEEE Software*, c/o Computer Science Dept., Oregon State University, Corvallis, OR 97331; (503) 754-3273; CSnet, lewis@oregon-state; Compmail+, t.lewis. Materials are due by **October 1, 1987**.

14th International Conference on Electric Contacts (IEEE): June 20-24, 1988, Paris. Submit an abstract (200 words maximum) by

 Conferences that the Computer Society sponsors or participates in are indicated by the Computer Society logo; additional conference sponsors are listed in parentheses. Other conferences of interest to our readers are also included.

For inclusion in Call for Papers or Calendar, submit information **six weeks before the month of publication** (e.g., for the **November 1987** issue, send information for receipt by **September 15, 1987**) to Calendar Editor, *Computer*, 10662 Los Vaqueros Cir., Los Alamitos, CA 90720.

October 1, 1987, to S.E.E., 48 Rue de la Procession, 75724 Paris Cedex 15, France.

Ⓢ Symposium on the Engineering of Computer-Based Medical Systems: June 8-10, 1988, Minneapolis, Minnesota. Four copies of abstracts are sought by October 10, 1987, and four copies of manuscripts are required by December 2, 1987, addressed to Bart Galle, Continuing Medical Education, Box 202, University of Minnesota Medical Clinic, 420 Delaware St. SE, Minneapolis, MN 55455; (612) 626-5525.

CSC-88, 16th Annual Computer Science Conference (ACM): February 23-25, 1988, Atlanta. Submit five copies of papers by October 15, 1987, to Richard DeMillo, Software Engineering Research Center, Georgia Institute of Technology, Atlanta, GA 30332.

1988 International Computers in Engineering Conference: Real-World Applications of Expert Systems and Artificial Intelligence (ASME): August 7-11, 1988, San Francisco. Submit abstracts by October 15, 1987, to Edward M. Patton, US Army Ballistic Research Lab, Aberdeen Proving Grounds, MD 21005.

Fourth International Software Process Workshop (IEEE, ACM): May 11-13, 1988, Devon, England. Position papers (three pages maximum) are sought by October 16, 1987 to Leon Osterweil, Computer Science Dept., University of Colorado, Campus Box 430, Boulder, CO 80309; (303) 492-8787.

Ⓢ FTCS 18, 18th International Symposium on Fault-Tolerant Computing: June 27-30, 1988, Tokyo. Six copies of abstracts are sought before October 20, 1987, and six copies of manuscripts should be submitted by November 15, 1987, all addressed to FTCS 18 Program Committee, PO Box 151, Hiroshima Central Post Office, Hiroshima 730-91, Japan.

Ⓢ 18th International Symposium on Multiple-Valued Logic: May 24-26, 1988, Palma de Mallorca, Spain. Five copies of manuscripts, limited to 20 doubled-spaced typewritten pages (including 50 to 100-word abstract), are sought by November 1, 1987 to Charles B. Silio, Electrical Engineering Dept., University of Maryland, College Park, MD 20742, for the Americas; to C. Moraga, University of Dortmund, FB Informatik, Postfach 500500, 4600 Dortmund 50, FRG, for Europe/Africa, or M. Mukaidono, Faculty of Engineering, Meiji University, 1-1-1 Higashi-mita Tama-ku, Kawasaki-shi 214, Japan, for Asia/Pacific.

Ⓢ IEEE Transactions on Computers: Papers are sought for a special issue on architectural support for programming languages and operating systems. Guidelines for submitting manuscripts appear in every issue of *IEEE Transactions on Computers*. Send seven copies of the manuscript by November 1, 1987, to Randy Katz, Dept. of Electrical Engineering and Computer Science, Computer Science Division, Evans Hall, University of California, Berkeley, CA 94720; (415) 642-8778.

RATES: \$12 per line, \$120 minimum charge (up to ten lines). Average six typeset words per line, nine lines per column inch. Add \$10 for box number. Send copy at least six weeks prior to month of publication to: Hedi Rex or Marlan Tibayan, Classified Advertising, *COMPUTER Magazine*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.

In order to conform to the Age Discrimination in Employment Act and to discourage age discrimination, *COMPUTER* may reject any advertisement containing any of these phrases or similar ones: "... recent college grads. . .," "... 1-4 years maximum experience. . .," "... up to 5 years experience. . .," or "... 10 years maximum experience." *COMPUTER* reserves the right to append to any advertisement, without specific notice to the advertiser, "Experience ranges are suggested minimum requirements, not maximums." *COMPUTER* assumes that, since advertisers have been notified of this policy in advance, they agree that any experience requirements, whether stated as ranges or otherwise, will be construed by the reader as minimum requirements only.

NORTHERN ARIZONA UNIVERSITY

Computer Science, Assist./Assoc. Prof., hardware or software. Hardware position requires excellence in digital design, computer architecture, real time systems, microprocessor applications. Software position requires excellence in all areas of computer software including automata theory, data structures, compiler design and theory. The search will remain open until filled. Screening Committee will review applications as they arrive. Forward letter of application, resume, 3 references to: Dean, College of Engineering & Technology, NAU, Box 15600, Flagstaff, AZ 86011. For information call 602-523-2880. NAU is an Equal Opportunity Affirmative Action Institution. Women and minorities are encouraged to apply.

NAVAL POSTGRADUATE SCHOOL Monterey, CA

Position Announcement in Computer Science

The Department of Computer Science has immediate openings for faculty positions at all levels in all areas. An applicant should have a PhD in Computer Science or a related field and have a strong interest in both graduate teaching and research. Senior applicants must have distinguished research records. Appointments can begin at any time during the year.

The Department offers MS and PhD degrees in Computer Science supported by well-equipped instructional/research facilities and a full-time technical staff. The faculty normally teach for two quarters and conduct full-time research during the other two quarters.

Please send a detailed resume and three letters of reference to: Search Committee, Computer Science Department (Code 52), Naval Postgraduate School, Monterey, CA 93943, tel. # (408) 646-2449. NPS IS AN EQUAL OPPORTUNITY/AFFIRMATIVE ACTION EMPLOYER.

CAREER

PHILIPS INFORMATION SYSTEMS LIMITED Montreal, Canada

The Research and Development Department of Philips Information Systems has immediate openings in the field of User Interface Design applied to advanced Office Automation applications. Applicants must have relevant experience, and/or have completed advanced studies. A background which combines computer science and psychology would be an asset. Applications for both regular and visiting positions will be entertained.

The R&D Department is involved in the development of sophisticated, highly integrated applications for Office Automation, including desktop publishing, communications, and database management.

Philips Information Systems Limited is part of N.V. Philips' Gloeilampenfabrieken, a (US\$16 Billion) multinational corporation based in the Netherlands, employing 344,000 people worldwide. The Montreal International Development Center shares the Office Automation mandate with other IDCs in Europe, and is thus part of a large R&D organization, with access to a wide range of resources.

Montreal is a cosmopolitan city with excellent facilities for cultural, educational, entertainment and sports activities. Close to half of the city's 2 million people are English-speaking, and most services are available in both English and French.

Preference will be given to Canadian citizens and permanent residents of Canada, in accordance with Canadian Immigration regulations. Applicants should send their resume to:

Dr. Y.-s. Kwong
Vice-President, Research & Development
Philips Information Systems Limited
600 Dr. Frederik Philips Blvd.
St. Laurent, Quebec, Canada
H4M 2S9

FLORIDA STATE UNIVERSITY Department of Computer Science

Applications are invited for tenure track faculty positions at all levels in the Department of Computer Science starting August 1988. Areas of special interest include, but are not limited to, operating systems, programming languages, computer graphics, software engineering, and computer architecture.

Our department is a young and rapidly growing one. It offers B.S., M.S., and Ph.D. degrees, and it places a strong emphasis on excellence in research and teaching. Research facilities include a supercomputer CYBER 205, a CYBER 760, CYBER 730s, a graphics lab, and a local network consisting of a VAX 11/780, a VAX 11/750, several SUN workstations, and Symbolics 3640. Applicants should have a Ph.D. in computer science or a closely related area, and a strong commitment to teaching and research. Send resume and names of at least three professional references to:

Dr. Abe Kandel, Chairman
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4019

Florida State University is an Equal Opportunity Affirmative Action Employer.

OPPORTUNITIES

STATE UNIVERSITY OF NEW YORK AT BUFFALO

The Department of Computer Science is seeking faculty at all levels, Assistant, Associate and full Professor, to begin Fall, 1988. Applicants at the junior level must have a Ph.D. in Computer Science, or a related field, and superior research ability. Applicants at the senior level must have a proven research and funding record.

The University at Buffalo is the largest public university in New York and New England, with approximately 26,000 students enrolled in 93 undergraduate, 101 master's and 94 doctoral programs, including law, medical, and business schools.

Present research areas include artificial intelligence, computer vision, numerical analysis, parallel algorithms, theory of computation, VLSI, etc., with an annual research expenditure of \$1 million. The Department offers BA, BS, MS, and PhD degrees, with controlled undergraduate enrollment. Departmental computing facilities include a VAX 11/785, two VAX 11/750s, twelve SUNs, seven lisp machines and several image processing/graphics systems. The current department consists of 14 tenure track faculty, 3 lecturers, 3 full-time technical staff and over 60 supported graduate students. The Department is expanding, with additional faculty lines committed annually. Salaries are extremely competitive. Resumes and names of four references should be directed to: Prof. Sargur N. Srihari, Acting Chairman, Department of Computer Science, 226 Bell Hall, SUNY at Buffalo, Buffalo, NY 14260 (CSnet: srihari@buffalo). For full consideration, applications should be received by February 1, 1988. SUNY is an affirmative action/equal opportunity employer.

UNIVERSITY OF AUCKLAND NEW ZEALAND CHAIR IN COMPUTER SCIENCE

The Department of Computer Science at The University of Auckland invites applications for a newly established second Chair.

Applicants should be qualified Computer Scientists whose personal qualities and experience will enable them to contribute significantly to the continuing development of Computer Science at Auckland both in teaching and research. As the successful candidate will be expected to play a leading role in the promotion of research within the department, a demonstrated history of research administration and liaison with industry and external agencies is expected in addition to an accomplished research record.

Applicants should be prepared to share the department administrative load including, eventually, periods as Head of the Department. Commencing salary will be established in the range NZ\$68,000-85,000 per annum, having regard for qualifications and experience of the candidate concerned.

Further information including Conditions of Appointment and Method of Application are available from the Assistant Registrar (Academic Appointments), University of Auckland. Applications should be forwarded before the closing date of 1 OCTOBER 1987.

W.B. Nicoll, REGISTRAR, University of Auckland, Private Bag, Auckland, NEW ZEALAND.

CLAREMONT McKENNA COLLEGE Endowed Position In Computer Science and Applied Mathematics

Applications are invited for an endowed tenure-track position in Computer Science and Applied Mathematics with rank and salary dependent on qualifications. Starting date fall 1988.

Claremont McKenna College is a liberal arts college with 800 students. It is a member of the Claremont Colleges (along with Pomona, Scripps, Harvey Mudd, and Pitzer Colleges and Claremont Graduate School). The Claremont Colleges have a total of forty-three mathematicians and computer scientists, and are located in Claremont, Southern California.

Qualifications for the position include a Ph.D. in a computer-related field such as Computer Science, Mathematics, Operations Research, or Information Science. If the degree is in a field other than Computer Science, substantial formal education in Computer Science is required.

Applicants should have a strong commitment to undergraduate teaching, an established scholarly record, and practical experience with computer applications. The appointee will be expected to teach some applied mathematics courses in addition to computer science courses and to participate in course and program development.

The College is an equal opportunity/affirmative action employer. Applications will be reviewed as soon as received and a decision reached preferably by January 1988. Please send resume and the names of four references to Professor John Ferling, Chairman, Computer Science Search Committee, c/o Dean of Faculty's Office, Claremont McKenna College, Claremont, CA 91711.

SOFTWARE ENGINEER

Software Engineer needed to develop Law Enforcement Software System, require M.S. Degree in Computer Science, knowledge of Unify Relational Data Base, ability to work with various hardware, such as PC's, AT&T, Altos, Wicat and HP computers, advanced proficiency in C language programming, experience with UNIX and DOS. Forty-eight hour week. Salary \$28,600 per year. Job Order #73546. Contact Utah Job Service, P.O. Box 307, Logan, Utah 84321.

SENIOR COMPUTER SCIENTIST

Develop advanced systems software, especially compilers and related CAIS tools. Develop new front-end for Ada compilers. Revise and compare performance of algorithms for solving front-end problems, e.g., overload resolution in Ada. Analyze results of compiler's test runs. Identify and correct errors in the compiler.

Extensive knowledge of various programming languages required: (Ada, Modula-2, Pascal, C, SNOBOL 4 and Icon).

Thorough understanding of compiler organization and development (scanning, parsing, semantic analysis, internal representations and error reporting and recovery) required.

Knowledge of the functionality and structure of parser generators. Ph.D. in Computer Science required. \$49,000 per year. Send resume to Job Service East, 6206 Broad Street, Pittsburgh, PA 15206. Refer to Job Order No. 3971023.

UNIVERSITY OF CALIFORNIA, IRVINE Department of Electrical Engineering Computer Engineering Area

The Department of Electrical Engineering at the University of California, Irvine, invites outstanding individuals to apply for tenure track positions at all levels in the area of computer engineering. Some positions will be filled by January, 1988 and others will be filled by June 30, 1988. The levels of these appointments may be at Assistant Professor, Associate Professor, or Professor depending on accomplishments and stature in the field. Subareas of special interest include, but are not limited to:

1. Advanced computer architectures e.g., distributed architectures, highly parallel architectures, ultra-reliable architectures, inference machine architectures, and applications-oriented architectures.

2. System software including operating systems.

3. Advanced applications of computer engineering including computer communications, artificial intelligence, etc.

Responsibilities include research and teaching at the graduate and undergraduate levels.

The Department has research activities in design automation, VLSI design, distributed computing, fault-tolerant computing, and real-time computing, and in closely related areas including digital signal processing, image understanding, pattern recognition, robotics, digital communication, and communication networks. The Department currently has nineteen faculty members, two of whom are members of the National Academy of Engineering and seven are Fellows of the IEEE.

Send your complete resume with a list of at least three references to Kane Kim, Chairman of the Computer Engineering Faculty Search Committee, Department of Electrical Engineering, University of California, Irvine, California 92717. The University of California is an affirmative action/equal opportunity employer. To be considered for the positions to be filled by January, 1988, apply before October 15, 1987. For the remaining positions, apply before February 15, 1988.

AUBURN UNIVERSITY Computer Science & Engineering Department

Applications are invited for tenure-track faculty positions at the Associate Professor level. Applications at the Assistant Professor level will also be considered. A Ph.D. in Computer Science, Computer Engineering, Electrical Engineering or a closely related discipline is required. Preferred areas of interest include Data Base Systems, Computer Architecture, and Operating Systems. Positions are open for the 87/88 Academic year. Salary and faculty rank are competitive depending upon experience and qualifications. Responsibilities include teaching at the Graduate and Undergraduate levels and building the ongoing research program. The department offers Bachelor's degrees in Computer Science and Computer Engineering as well as the M.S. and Ph.D. An outstanding research program has been established in the above disciplines. Applicants should submit a detailed resume and three references to Charles R. Vick, Department Head, Computer Science and Engineering, Auburn University, AL 36849-5347. Women and minorities are encouraged to apply. Auburn University is an Equal Opportunity/Affirmative Action Employer.

BOOK REVIEWS

Editor: Wiley McKinzie, School of Computer Science and Technology, Rochester Institute of Technology,
Rochester, NY 14623; Compmail, w.mckinzie; CSnet, wrm@rit

The Design of the Unix Operating System

Maurice J. Bach (Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986, 471 pp., \$31.95)

For years, the only publicly available description of the Unix operating system was the *Lyon's Book*, produced by the University of New South Wales. It contained a complete listing of the Unix source code plus a line-by-line commentary, and it was available to people without source licenses. Unfortunately, it only described Version 6 Unix, which by now is at least four versions out of date. Those of us who were lucky enough to have source licenses were at least able to see the source code, but it was not very understandable in the absence of a commentary.

Maurice Bach has written a book that at least alleviates our problems. Based on a course that he taught at AT&T Bell Laboratories in 1983 and 1984, the book is designed as a companion to the source code. The book explains the algorithms involved inside the kernel rather than providing a line-by-line commentary on the code. It is up-to-date in that it describes Unix System V Release 2, which is the most common AT&T variant of Unix used today. To add generality to the book, Bach also covers some features from Unix System V Release 3 and from the Berkeley Software Distributions (BSD) versions of Unix.

The book is a treasure trove of information. The first chapter consists of an overview of the Unix system for those people who are not familiar with it. The next chapter previews the rest of the book by describing first the structure of the kernel and then the process subsystem and the file subsystem. The rest of the book is divided into three sections. The first one is devoted to the file subsystem, with chapters on the buffer cache, the representation of files and the system call interface to the file sys-

tem. The next section covers the process subsystem, including memory management, interprocess communications and the I/O subsystem. (A discussion of the I/O subsystem appears here instead of in the file subsystem section, where it might more logically belong. This arrangement allows Bach to discuss the issues of process control that accompany the writing of terminal drivers.) The final two chapters cover advanced topics—multiprocessor Unix systems and distributed Unix systems.

Before reading the book seriously, I skimmed it to try to get a feel for it. I found the presentation very pleasing. Bach has broken out the algorithms and presents them in a very readable pseudocode. He provides a lot of pictures describing how data structures fit together. At this first glance, the book seemed very readable. I was not disappointed.

After glancing over the table of contents, I wanted to see how good the coverage of the particular topics is. I have spent a lot of time writing device drivers and protocols, usually implemented as line disciplines, at least since the advent of System III. Unfortunately, none of the information that accompanied the release of System III said anything about the line disciplines other than that the entries for the routines belonged in the *linesw* table. Turning to the section on terminal drivers, the first thing that I saw was a bullet list of seven items describing what a line discipline had to accomplish. Here, already, was more documentation than AT&T had provided on the topic. The description is clear and succinct. It includes a step-by-step description of how to put characters on and take them off *clists*. For anyone who has ever spent time figuring out exactly how to manipulate all the pointers and counts, it's nice to see this procedure written down clearly.

From line disciplines, I went to an area that I was not so familiar with—*streams*. Streams are a new feature in System V, release 3 that were designed by Dennis Ritchie to provide more flexibility and modularity in the I/O subsystem. Bach provides a good description

of what streams are and how they are intended to be used. He also provides a high-level description of how to use streams to implement windows on a terminal. This description draws on a paper that Pike wrote describing the Blit terminal. Bach concludes the section on streams with a discussion of why Ritchie implemented them the way that he did. In this discussion, he identifies some of the remaining shortcomings of the current implementation.

Bach's chapter on interprocess communications covers three distinct areas. The first is the use of the *ptrace* system call as a form of interprocess communications for debugging. He then discusses all of the System V IPC mechanisms, message queues, shared memory and semaphores as a group, due to their great similarity in interface. Lastly, he talks about network communications by concentrating on the Berkeley socket implementation.

The list of topics discussed can go on and on. As I stated earlier, the book is a real storehouse of information on Unix in particular and operating systems in general. Bach has culled out the essential pieces of Unix by describing the algorithms and data structures in a pseudocode that is easy to understand. He presents a very balanced view of Unix, often describing the shortcomings of the various features he discusses.

This book belongs on the shelf of everyone who is involved with the Unix operating system on a regular basis. Having it, you will have a much better understanding of how the system works, and therefore, how to make better use of the system. It is also useful for anyone who wants to know about operating systems. While designed solely for a course on Unix, it describes the major functions and features of operating systems, so that it could also be used for a general operating systems course.

The only complaint I have, and considering the purpose of the book it may not be a valid one, is the lack of discussion of Berkeley extensions to the Unix system. Berkeley has made extensions to Unix in three areas: the fast file system, the terminal handlers in the line discipline area, and the use of sockets for network communications. Of the three, only the last is covered to any extent. It would have been nice if the book covered the other two topics in the appropriate chapters, since no easily readable discussion of them is available anywhere.

Lorne H. Schachter
Bell Communications Research

Data Networks

Dimitri Bertsekas and Robert Gallager (Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987, 486 pp., \$37.33)

This book was written as a textbook in computer networking for senior undergraduates and masters level graduate students in computer networking. The teaching experience of the authors is evident in the careful development of topics in convenient, lesson-sized installments. Each chapter includes numerous problems to be worked out by the student. The authors have been careful to make these problems reasonable for the student to solve yet nontrivial in their relationship to real-world networking problems.

The book is equally suitable as a reference for those already in the field. It is well organized in the material it covers, and the index appropriately guides the reader to the relevant coverage. I would not, however, recommend this book to someone who is not already familiar with the issues and technology of computer networking. Some of the definitions used conflict with common usage. For example, the term *virtual circuit* is used in the old network-routing algorithms context rather than the higher level connection-oriented protocol context. While not a fatal flaw, such usage could confuse the naive reader.

More important than the misleading use of a few terms is the formal and stilted presentation style. The book is heavy reading. A reader unfamiliar with the jargon and acronyms of networking could find it next to impossible to comprehend. I often found myself reading passages several times in order to understand the authors' intent.

The difficult presentation style is doubly unfortunate, as the book otherwise is excellent. The authors set out to write a book which covered both the theoretical aspects of networking and the practical implementation of those theories. They succeeded on both counts and provide excellent coverage of many of the issues in low-level network protocol design, particularly those affecting performance. From queueing theory to optimal routing strategies, they develop the theory behind many of the challenges facing network designers, then build on that theory to show how the problems discussed are solved in real networks today.

The first 100 pages provide general coverage of basic networking: layering protocols, the International Standards Organization's (ISO) reference model

for Open Systems Interconnection (OSI), physical aspects of communications media, error detection, error recovery, and framing. The remaining 350 pages consist of four chapters, one each on delay models, multiaccess communications, routing, and flow control. The depth of coverage is enhanced by the extensive, annotated references which appear at the end of each chapter. The following references for Section 4.2 are typical: "The Aloha network is first described in [ROB72]. The problem of stability was discussed in [MET 73], [Lak75], and [CaH75]. Binary exponential back off was developed in [MEB76]. Modern approaches to stability are treated in [HVL83] and [RIV85]." In all, the bibliography includes over 400 articles and books.

The total audience for this book is limited by the book's coverage of only the lower layers of the ISO OSI network architecture model. Given the profusion of standards available for these network layers, few practitioners need the depth of coverage provided by this book. However, for those interested in getting a better understanding of technical networking issues and an appreciation of the considerations that go into developing protocols at the lower networking layers, this book is right on target.

Vincent C. Jones

Big Blue: IBM's Use and Abuse of Power

Richard Thomas DeLamarter (Dodd Mead & Company, New York, 1986, 393 pp., \$22.95)

If the purpose of a book review is to leave the review reader with a desire either to read or to ignore a book, then this review will likely meet its objective. The title of the book tells the whole story. If you hate IBM (big business, the defense budget, the Republican Party, etc.), then you will love this book. If you do not hate IBM (or any of the preceding), then you will probably dislike this book.

Even ignoring the title, the author's thesis is clear from the onset. In January, 1974, he was hired as an economist by the Department of Justice's Antitrust Division. Eight years later, he departed when the (Republican) government declared the case against IBM to be without merit. Interestingly, I too was hired at that time, but by the accused, with whom I spent the next seven years in various development and marketing positions. However, my perspective on the issues raised by the author, most of which are entirely

legitimate, leads me to entirely different conclusions.

The first chapter, entitled "The Education of T.J. Watson," chronicles Watson's early days at NCR, which he left after he and other company executives were successfully prosecuted under the newly legislated antitrust laws. Throughout the book, DeLamarter incessantly refers to that period of Watson's career as his training in (monopolistic) crime by NCR's president, John Patterson. Other historians have positively characterized that period as the basis for IBM's Business Conduct Policies, the strictly enforced guidelines governing the business behavior of all IBM employees.

It is this perspective of the author that makes the book both interesting and irritating. Certainly, cynical analysts of IBM (who probably have never been employed by the company) imply that those policy guidelines are more often breached than followed. However, IBM employees know the serious consequences of such a breach. In fact, those guidelines which were so ingrained in me by my former employer underlie my own assessment of the clearly articulated arguments presented in this book.

DeLamarter leads the reader through a history of IBM products, including Hollerith's tabulating machines, unit record equipment, the first 700 series of computers, the System 360, and the System 370. His analyses of IBM mainframe, memory, and peripheral product announcement schedules and pricing policies obviously reflect the well considered and predetermined conclusions of the government's costly eight-year effort. On a page-by-page basis, I felt the compulsion to rebut those conclusions in this review.

Of particular interest to me were the conclusions drawn with respect to IBM's product policies in the mid- to late 1960's. Unlike many other computer manufacturers, IBM had no viable interactive (timesharing) product. The author contends that to avoid losing market share, IBM deliberately announced System 360-based timesharing products which were undeliverable, or did not work.

It is certainly true that IBM missed the boat in the original S360 design by not realizing the emerging importance of timesharing. But the author's simplistic conclusion (founded in his obvious predilection for seeing conspiracy under every rock) entirely ignores the technical difficulty of regressively altering this acknowledged state-of-the-art batch system in order to make it suitable for interactive use. Any IBM cus-

tomer who witnessed IBM's Herculean attempts to adapt those batch systems for interactive use, as I did as a graduate student at the end of the 1960's, will question the author's conclusions.

The book is weakened by two questionable implications. The first is the implication that IBM's competition simply had no chance against IBM's illegal practices, even when they could offer superior products. In 1967, as an employee of a large multinational timesharing company, I witnessed business practices against IBM that were at best questionable, but more likely illegal. This company, cited by the author as having a clearly superior timesharing offering, did not fail because of anything IBM did or did not do. This company failed because of weak management which was inexperienced in this new market. In those relatively early days of the computer industry, many large multinationals entered the market and withdrew a short time later. Were those failures all attributable to evildoing by IBM, or did other factors come into play?

The other questionable implication made by the author is that the marketplace had no freedom of choice in product selection. For a decade, beginning in the late 1950's, the field was wide open for both computer manufacturers and for customers. The nature of trade in the U.S. provides everyone with the freedom to choose any product or vendor. Why did so many new computer customers select IBM?

The author provides detailed analyses of IBM pricing which indicate that IBM enticed new customers with loss-leading, entry-level systems. He argues that IBM's strategy was to recoup those losses later by selling overpriced memory and peripherals to tied-in customers. In the absence of equivalent government-collected data to refute those analyses, my reaction to this admonishment is "So what?" IBM is not the first, and will certainly not be the last company to use loss leaders to attract business. And in any case, no customer is really an unwilling captive of any vendor. Company politics, followed by economics, usually determine the degree of capture felt by unsatisfied customers of any vendor. This principle is also true in the computer industry. The author portrays the computer marketplace as completely manipulable by IBM. This charge is clearly nonsense.

If you are predisposed to arguing with your colleagues about the evil monster, IBM, I strongly recommend that you study this book. It will provide you with substantial material for your assaults. On the other hand, if you are a

vehement capitalist with an inclination to write a book, counterarguments to DeLamarter's work will provide you with more than enough material.

Sorel Reisman
California State University, Fullerton

Handbook of Software Quality Assurance

G. Gordon Schulmeyer and James I. McManus (eds.) (Van Nostrand Reinhold Company, New York, 1987, 454 pp., \$32.95)

This handbook is a collection of 17 original papers on various aspects of software quality assurance (SQA), authored by 17 authorities in the field. And I do not use the term "authority" loosely.

The authors are all SQA professionals who deal with the implementation of software quality assurance on a daily basis. They include such noted authorities as Thomas J. McCabe ("Software Complexity Metric," *IEEE Software Engineering Transactions*, December, 1976), Robert H. Dunn (*Quality Assurance for Computer Software*, McGraw-Hill, 1982 (with Richard Ullman)), Matthew J. Fisher (*Software Quality Management*, Petrocelli Books, 1979 (with John D. Cooper)), and William E. Perry (*Effective Methods of EDP Quality Assurance*, Q.E.D. Information Sciences, 1981). In the Preface, the editors state that their objective was to produce a "collection of experiences and expectations of some of the best people in the software quality field," rather than a collection of stuffy theory written by philosophers. They have very definitely succeeded in accomplishing this objective.

The handbook is intended for use by the practicing SQA professional, but is suitable for use by the software professional needing assistance in producing a quality software product, since many of the essential aspects of the field are covered. It is logically divided in two sections, with a total of 17 chapters and two appendices.

The Preface contains a name index, a subject index, and a precis of each chapter to allow the professional to select the desired information. Each chapter also supplies footnoted references, general references, or both, for additional research.

The first section of the handbook (Chapters 1-8) "sets the stage," introducing definitions, describing the interaction of the software quality function with other software specialty areas,

and providing an excellent overview of the government and industry standardization efforts, complete with outlines of the most current SQA standards and a table comparing the requirements of these different standards.

The second section (Chapters 9-17) focuses on tools, techniques, and methodologies for SQA implementation. This section includes topics such as design inspections, software configuration management, statistical methods applied to software (for software quality control), and internal standards development. In particular, Chapter 10 discusses the application of the Pareti Principle (concentrating on the vital few, not the trivial many) to SQA implementation, and Chapter 17 contains interesting and revealing results from a survey conducted by the Quality Assurance Institute of its members on software quality assurance. The chapters on software reliability and statistical methods contain some valuable new metrics.

However, do not misconstrue the comments above to imply that the non-software professional could use the handbook for self-instruction. The handbook does not contain a software development primer as many of the publications on SQA do. It is designed to provide hints and experiences for SQA personnel and for those non-specialists who are interested in the subject and are already familiar with software development methodologies. Since it is intended as a handbook and is written by a diverse group of professionals, some discontinuity occurs in the terms used (*SQAM* vs. *SQA* vs. *SQC*), and some personal bias is injected ("Over the years many individuals who possess a degree in education or the liberal arts have made the switch to software. . . They do, however, make poor SQA engineers.").

Despite these minor problems, the handbook is sufficiently self-contained, well organized, and interesting to read. Although it is not a primer on software quality, I plan to use it as a resource for a graduate level course I recently designed in Software Quality Assurance Engineering (Amber University, Garland, Texas). In addition, I have found a number of useful techniques in the handbook and intend to keep it handy.

I feel that this handbook is definitely a worthwhile addition to the library of any SQA professional—he or she can expect to gain from the expertise and experience presented.

Vincent L. Boyer, Sr.
E-Systems, Inc., Garland Division

NEW LITERATURE

Frost & Sullivan reports. According to *Advanced Motion Controls in Europe* (#E920, 348 pp., \$2700), robotics is the fastest growing use of advanced motion controls in Europe, following the trend set in Japan. The company says that the rapid spread of automated production systems in factories is the industrial change that will most mark the future.

The Optoelectronic Components Market in the US (#A1663, 234 pp., \$1950) forecasts strong growth for optical ICs because of their tremendous potential for future uses.

Data Communications in the Industrial Environment (#E900, 353 pp., \$2750) predicts increasing European investments in automation and computer systems over the next few years despite frequent start-up problems and unexpected expenses. It emphasizes that markets for industrial data communication products depend on progress in developing standards to make products compatible.

Customer Service, Frost & Sullivan, 106 Fulton St., NY 10038; (212) 233-1080 or Customer Service, Frost & Sullivan, Sullivan House, 4 Grosvenor Gardens, London SW1W 0DH; 01-730-3438.

International Planning Information reports. IP1 offers a series of reports published by other companies. *Fiber Optics Outlook* (Mar. 1987, 114 pp., \$495) analyzes worldwide markets for fiber optic systems and components, with forecasts to the year 2001. Published by Futurecast Learning Center, the report focuses on issues important to users. Extra copies cost \$100.

A monthly summary from Semiconductor Industry Reports called *Semiconductor Industry Update* (\$345/year) abstracts news from semiconductor trade publications.

International Planning Information, 465 Convention Way, Suite 1, Redwood City, CA 94063; (415) 364-9040.

Sensors. A Battelle multiclient program resulted in the report "Sensors: Miniaturization and Integration." The report identifies key development trends in sensor technology.

Program results are available to interested companies for DM 13,000. An extended version is planned, in English, for DM 30,000, that will include data on Japanese and US markets.

Barbara Soloway, Battelle, 505 King Ave., Columbus, OH 43201-2693.

Software productivity. The Summer 1987 *Guide to Software Productivity Aids* is a semiannual directory of software products and tools that can help improve productivity of systems development. It includes listings of 1221 software products with name, vendor, price, hardware/software requirements, and description.

Products are listed under 24 classifications, including application development systems, compilers, data dictionaries, database management systems, programming analyzers/optimizers, screen generators, and so forth.

Two tutorial chapters cover increasing productivity in traditional system development and through the information center concept.

The guide is available on a subscription basis for \$150/year. Single copies cost \$95. Applied Computer Research, PO Box 9280, Phoenix, AZ 85068-9280; (602) 995-5929.

Ada. A monthly report called *Ada Strategies* helps government contractors, government procurement offices, government agencies, insurance and financial institutions, airlines, industrial manufacturers, and computer vendors integrate Ada into their organizations.

The information service shows how to structure a corporate Ada implementation plan, set up and run an Ada training program, and so forth. It also provides information on available training documentation, software and hardware, the commercial market for Ada, and case studies.

The report is researched and written by Ralph E. Crafts, an Ada consultant and president of Software Strategies & Tactics.

The charter price is \$225 US, \$265 outside North America.

Cutter Information, 1100 Massachusetts Ave., Arlington, MA 02174; (617) 648-8700.

The Scientific & Engineering Software Guide from Lifeboat Associates is a reference catalog to over 100 PC-based application and development software packages designed for the science and engineering community. Entries include descriptions of features and benefits, compatibility, systems requirements, and whether a demonstration disk is available. The guide also contains ordering information. It is available by contacting a Lifeboat account representative. Lifeboat Associates, 55 S. Broadway, Tarrytown, NY 10591; (914) 332-1875.

Information processing. *World Markets for Information Processing Products to 1996* covers the current status of the industry, future trends, breakthroughs, and opportunities. It includes market forecasts and revenue projections to 1996, a user wish list with suggested vendor strategies, an analysis of the top 20 companies, and the importance of repositioning to provide business solutions instead of technological systems. Price: \$1250. Arthur D. Little Decision Resources, Attn.: Jean Carbone, 17 Acorn Park, Cambridge, MA 02140; (617) 864-5770, ext. 4425.

Resource directory. The first edition of the *Computers and Computing Information Resources Directory* (Martin Connors, ed., 1987, ISBN 0-8103-2141-6, 1271 pp., \$160) covers eight major types of sources of information about computers and computing. It describes over 4000 live sources from worldwide associations and user groups to consultants and research organizations. It also profiles more than 1500 periodicals, directories, and abstracting and indexing services.

Areas of computer utilization covered include artificial intelligence, general data processing, CAD/CAM, programming languages, personal computers, robotics, telecommunications, management information systems, computer retailing, and office automation.

Entries are arranged by type of information source. Three indexes provide other methods of accessing the information.

Gale Research, Book Tower, Detroit, MI 48226; (313) 961-2242.

ADVERTISER INDEX

Advanced Micro Devices	Cover IV
BDM International, Inc.	70
CACI	1
Casio, Inc.	Cover II
Computer Society Membership	8-9
Computer Society Ombudsman	102
Fall Joint Computer Conference	12-15
Gould Inc.	16
ICCAD '87	116-117
ICCD '87	92-93
IEEE Press	84
Interactive Software Engineering, Inc.	97
Lockheed Missiles & Space Company	7
Measurex Automation Systems	42
Meta Software Corporation	58
National Business Employment Weekly	91
Northrop Corporation	69
Rapitech Systems Inc.	11
RIAO 88 Conference	41
Texas Instruments Inc.	5
University of California, Los Angeles	97
Classified Advertising	122-123

FOR DISPLAY ADVERTISING INFORMATION CONTACT

Northern California and Pacific Northwest: Don Farris Co., 161 W. 25th Ave., Ste. 102B, San Mateo, CA 94403; (415) 349-2222.
Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Ste. 100, Torrance, CA 90505; (213) 373-9604.
Southwest: The House Co., 3000 Wesleyan, Houston, TX 77027; (713) 622-2868.
Midwest: Knorr Micromedia, Inc., 333 N. Michigan Ave., Chicago, IL 60601; (312) 726-2633.
East Coast: Hart Associates, PO Box 339, 42 Lake Blvd., Matawan, NJ 07747; (201) 583-8500.
New England: Arpin Associates, PO Box 227, Weston, MA 02193; (617) 899-5613.
Southeast/Telemarketing: Kay Young & Assoc., 2081 Business Center Dr., Suite 180, Irvine, CA 92714; (714) 551-4924.
Europe: Heinz J. Gorgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); (0 21 53) 8 99 88.

Advertising Director: Dawn Peck.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

Computer, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-8380.

PRODUCT INDEX

	RS #	Page#
Boards		
Emulation board	40	109
Circuits		
IC Announcements	135-145	114
Components		
Microsystems Announ.	120-130	115
Conferences & Courses		
FJCC	—	12-15
ICCAD '87	—	116-117
ICCD '87	—	92-93
RIAO 88	—	41
Data Communications		
Communications package	42	109
Communications server	38	109
I/O Related Equipment		
Graphics printer	46	110
Other Products & Services		
Training course	32	107
Publishers		
Book	7	84
CS Membership	—	8-9
CS Ombudsman	—	102
Journal	2	91
Recruitment		
Classified Advertising	—	122-123
Director	—	97
Technical professional	—	7, 42, 69, 70, Cover IV
Software		
CASE	34	108
Data collection	52	112
DOS extensions	33	108
Graphic system	5	58
Language converter	3	11
Lisp	53, 57	112, 113
Memory expansion	59	113
Network connectivity	35, 41	108, 109
Structural engr. analysis	36	108
Systems		
Board-level computer	37	109
Computer	4	16
Development tool	58	113
Expert system	—	5
Image processing	43	110
Microcomputer	51, 55	112
Network simulation	—	1
Neurocomputer	31	107
OEM system	54	112
Slotcard computer system	39	109
Software tool	1, 6	Cover II, 97
Superminicomputer	30, 44, 45, 56	107, 110, 113
Workstation	47-50	110, 111



THE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Executive Committee

President: Roy L. Russo*
IBM T.J. Watson Research Center
PO Box 218, Route 134
Yorktown Heights, NY 10598
(914) 945-3085

President-Elect: Edward A. Parrish, Jr.*

Vice Presidents

Education: Michael C. Mulder (1st VP)*

Technical Activities: Kenneth R. Anderson (2nd VP)*

Area Activities: Willis K. King†

Conferences and Tutorials: James H. Aylor†

Membership and Information: Merlin G. Smith†

Publications: J. T. Cain†

Standards: Helen M. Wood

Secretary: Duncan H. Lawrie

Treasurer: Joseph F. Urban

Purpose

The Computer Society strives to advance the theory and practice of computer science and engineering. It promotes the exchange of technical information among its 90,000 members around the world, and provides a wide range of services which are available to both members and nonmembers.

Membership

Members receive the highly acclaimed monthly magazine *Computer*, discounts on all society publications, discounts to attend conferences, and opportunities to serve in various capacities. Membership is open to members, associate members, and student members of the IEEE, and to non-IEEE members who qualify as affiliate members of the Computer Society.

Publications

Periodicals. The society publishes six magazines (*Computer*, *IEEE Computer Graphics and Applications*, *IEEE Design & Test of Computers*, *IEEE Expert*, *IEEE Micro*, *IEEE Software*) and three research

COMPUTER

September 1987 issue

(Card void after March 1988)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ ZIP _____

Country _____ Phone (____) _____

Please send

(Circle those you want):

- 201 Publications catalog
- 202 Membership information
- 203 Student membership information
- 204 Application for senior membership

Reader interest

(Circle what you liked,
add comments on the back)

- A Hypertext
- B Software Productivity
- C RISC Processors
- D Formal Program Design
- E Complex Instruction Sets
- F President's Message
- G Editor's Message
- H Standards
- J Update
- K Computer Society News
- L New Product Reviews
- M New Products
- N IC Announcements
- P Microsystem Announcements
- Q Calendar
- R Call for Papers
- S Book Reviews
- T New Literature

Product inquiries

(circle the numbers for products and advertisers
you want more information on)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	190
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

2

COMPUTER

September 1987 issue

(Card void after March 1988)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ ZIP _____

Country _____ Phone (____) _____

Please send

(Circle those you want):

- 201 Publications catalog
- 202 Membership information
- 203 Student membership information
- 204 Application for senior membership

Reader interest

(Circle what you liked,
add comments on the back)

- A Hypertext
- B Software Productivity
- C RISC Processors
- D Formal Program Design
- E Complex Instruction Sets
- F President's Message
- G Editor's Message
- H Standards
- J Update
- K Computer Society News
- L New Product Reviews
- M New Products
- N IC Announcements
- P Microsystem Announcements
- Q Calendar
- R Call for Papers
- S Book Reviews
- T New Literature

Product inquiries

(circle the numbers for products and advertisers
you want more information on)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	190
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

3

ADVERTISER INDEX

Advanced Micro Devices	Cover IV
BDM International, Inc.	70
CACI	1
Casio, Inc.	Cover II
Computer Society Membership	8-9
Computer Society Ombudsman	102
Fall 1988 Computer Conference	12-15
Go!	16

Editorial comments

Go!

ICC

IC/

IE

Ir

L

I

I liked: _____

I disliked: _____

I would like to see: _____

For reader service inquiries, use other side

PRODUCT INDEX

	RS #	Page#
Boards		
Emulation board	40	109
Circuits		
IC Announcements	135-145	114
Components		
Microsystems Announ.	120-130	115
Conferences & Courses		
FJCC	—	12-15
ICCAD '87	—	116-117
ICCD '87	—	92-93
RIAO 88	—	41

PO box address for
reader service cards only

PLACE
STAMP
HERE

**Computer
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA**

Editorial comments

I liked: _____

I disliked: _____

I would like to see: _____

For reader service inquiries, use other side

PO box address for
reader service cards only

PLACE
STAMP
HERE

**Computer
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA**



THE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Executive Committee

President: Roy L. Russo*
IBM T.J. Watson Research Center
PO Box 218, Route 134
Yorktown Heights, NY 10598
(914) 945-3085

President-Elect: Edward A. Parrish, Jr.*

Vice Presidents

Education: Michael C. Mulder (1st VP)*

Technical Activities: Kenneth R. Anderson (2nd VP)*

Area Activities: Willis K. King†

Conferences and Tutorials: James H. Aylor†

Membership and Information: Merlin G. Smith†

Publications: J.T. Cain†

Standards: Helen M. Wood

Secretary: Duncan H. Lawrie

Treasurer: Joseph E. Urban

Past President: Martha Sloan*

Division V Director: Martha Sloan

Division VIII Director: H. Troy Nagle†

Executive Director: T. Michael Elliott†

*voting member of the Board of Governors

†nonvoting member of the Board of Governors

Board of Governors

Term Ending 1987

Barry W. Boehm
Paul L. Borrill
Glen G. Langdon, Jr.
Duncan H. Lawrie
Susan L. Rosenbaum
Bruce D. Shriver
Harold S. Stone
Wing N. Toy
Helen M. Wood
Akihiko Yamada
Oscar N. Garcia†

Term Ending 1988

Mario R. Barbacci
Victor R. Basili
Lorraine M. Duvall
Michael Evangelist
Allen L. Hankinson
Laurel Kaleda
Ted Lewis
Ming T. Liu
Earl E. Swartzlander, Jr.
Joseph E. Urban

Next Board Meeting

8:30 a.m.-5 p.m., October 30, 1987,
Loew's Anatole Hotel, Dallas

Senior Staff

Executive Director: T. Michael Elliott
Editor and Publisher: True Seaborn

Director, Computer Society Press: Chip G. Stockton

Director, Conferences: William R. Habingreither

Director, Finance and Administration: Mary Ellen Curto

Offices of the Computer Society

Headquarters Office

1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone: (202) 371-0101

Telex: 7108250437 IEEE COMPSO

Publications Office

10662 Los Vaqueros Circle
Los Alamitos, CA 90720

Membership and General Information: (714) 821-8380

Publications Orders: (800) 272-6657

European Office

13, Avenue de l'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Telex: 25387 AVVALB

Purpose

The Computer Society strives to advance the theory and practice of computer science and engineering. It promotes the exchange of technical information among its 90,000 members around the world, and provides a wide range of services which are available to both members and nonmembers.

Membership

Members receive the highly acclaimed monthly magazine *Computer*, discounts on all society publications, discounts to attend conferences, and opportunities to serve in various capacities. Membership is open to members, associate members, and student members of the IEEE, and to non-IEEE members who qualify as affiliate members of the Computer Society.

Publications

Periodicals. The society publishes six magazines (*Computer*, *IEEE Computer Graphics and Applications*, *IEEE Design & Test of Computers*, *IEEE Expert*, *IEEE Micro*, *IEEE Software*) and three research publications (*IEEE Transactions on Computers*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Software Engineering*).

Conference Proceedings, Tutorial Texts, Standards Documents. The society publishes more than 100 new titles every year.

Computer. Received by all society members, *Computer* is an authoritative, easy-to-read monthly magazine containing tutorial, survey, and in-depth technical articles across the breadth of the computer field. Departments contain general and Computer Society news, conference coverage and calendar, interviews, new product and book reviews, etc.

All publications are available to members, nonmembers, libraries, and organizations.

Activities

Chapters. Over 100 regular and over 100 student chapters around the world provide the opportunity to interact with local colleagues, hear experts discuss technical issues, and serve the local professional community.

Technical Committees. Over 30 TCs provide the opportunity to interact with peers in technical specialty areas, receive newsletters, conduct conferences, tutorials, etc.

Standards Working Groups. Draft standards are written by over 60 SWGs in all areas of computer technology; after approval via vote, they become IEEE standards used throughout the industrial world.

Conferences/Educational Activities. The society holds about 100 conferences each year around the world and sponsors many educational activities, including computing sciences accreditation.

European Office

This office processes Computer Society membership applications and handles publication orders. Payments are accepted by cheques in Belgian francs, British pounds sterling, German marks, Swiss francs, or US dollars, or by American Express, Eurocard, MasterCard, or Visa credit cards.

Ombudsman

Members experiencing problems — late magazines, membership status problems, no answer to complaints — may write to the ombudsman at the Publications Office.

Information

Use the Reader Service Card to obtain the following material:

- Membership information and application (RS #202)
- Publications catalog (proceedings, tutorials, standards) (RS #201)
- Periodicals subscription application/information for individuals (members, sister-society members, others) (RS #200)
- Periodicals subscription application/information for organizations (libraries, companies, etc.) (RS #199)
- List of awards and award nomination forms (RS #198)
- Technical committee list and membership application (RS #197)
- Directory of officers, board members, committee chairs, representatives, staff, chapters, standards working groups, etc. (RS #196)

Some will follow the old standards. Some will create the new.

An engineer has only a few precious opportunities to create a new industry standard. This is one of them. If you're traveling within the limits of today's technology, but want to break through to the next generation of microprocessing, AMD has some exciting news for you... **Your appointment with destiny has arrived.**

AMD has gone way beyond the limits of existing microprocessing to create the super chip of the 1990's — the Am29000. At 32 bits and 17 MIPS, the Am29000 RISC based microprocessor is 400% faster than the current industry standard. This exciting evolutionary breakthrough will make a whole new range of applications possible; bringing the power of a main-frame to desktop workstations; and providing embedded controller applications such as fiber optic networking and laser printer and communications controllers.

Current openings exist for experienced hardware and software engineers and other high tech professionals who want their talent and creativity to spark the next generation of technology.

Join AMD. Get ready to make history.

Am29000

Product Planning Engineers

Join the team that will author the future generations of the new industry standard — the Am29000. Blend into our superb talent bank your experience in computer architecture, definition and design, along with your strength in C and UNIX and excellent writing skills. Bring along any experience in RISC architecture, I/O controllers, cache memory, or development software such as compilers, assemblers, debuggers, or hardware and software implementation of floating point arithmetic, and you are a prime candidate to be involved in all phases of defining and introducing products, including product definition, market research and applications.



PAL Design/ Product Planning Engineers

Opportunities are available to design high-speed 24 pin PAL circuits and product planning of next generation technologies from a customer application viewpoint.

Manager Software Applications

Lead 10-15 engineers providing support for our AMD processors. Take responsibility for definition, development, testing and maintenance of our software products which include O/S, languages, tools, and utilities as well as application programs. You need experience covering S/W development, contract administration, spec generation of microprocessor support S/W and some S/W management.

Software Support Engineers

Experience in UNIX and C, plus MS DOS, PASCAL, FORTRAN, and assembler makes you an ideal candidate. Be involved in customer support, developing tools and utilities, testing S/W packages, user manual review, documentation, writing application notes and magazine articles.

Contracts Administrator

Negotiate, monitor and oversee OEM and development contracts. Experience in administering S/W development contracts required.

Senior H/W-S/W Applications Engineers

Support the design-in of the Am29000 using your experience in H/W and S/W design of digital systems in computers and imbedded controllers. System architecture, strong communication skills and strength in H/W and S/W system performance optimization are a big plus.

Software QA Applications Engineers

Develop test plans and perform S/W QA testing, develop and implement test suites for O/S, compilers and assemblers and develop tools and utilities to aid in testing.

Applications Engineers

Join in the design and support of Am29000 demo/evaluation boards and systems, do customer support and publish documentation. You will need experience in H/W design of digital systems and familiarity with assembler, C, UNIX, and MS DOS.

Documentation Specialist

Maintain the documentation for Am29000 support tools as well as generate documentation for user and reference manuals for new tools. Technical writing with emphasis on S/W tools such as compilers, assemblers, debuggers is required.

AMD is writing a new chapter in the history of technology for those who dream of creating the new. You see, at AMD we figure that, **if you don't make history, you're history.**

For immediate consideration, please call (408) 749-3119 or send your resume to Advanced Micro Devices, Professional Employment, MS-57, 901 Thompson Place, P.O. Box 3453, Sunnyvale, CA 94088. An equal opportunity employer.

